DIVA Architectural Perspectives on Information Visualization

Schonhage, S.P.C.

2001

document version
Publisher's PDF, also known as Version of record

DIVA
ARCHITECTURAL PERSPECTIVES ON INFORMATION VISUALIZATION

BASTIAAN SCHÖNHAGE

SIKS Dissertation Series No. 2001-7.

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.

Promotiecommissie:
prof.dr. J.C. van Vliet (promotor)
dr. A. Eliëns (co-promotor)
prof.dr. M. Jern (Linköping University Sweden, Advanced Visual Systems)
prof.dr. J. Bosch (Universiteit Groningen)
prof.dr.ir. H.E. Bal (Vrije Universiteit Amsterdam)
dr.ir. P.W.P.J. Grefen (Universiteit Twente)

VRIJE UNIVERSITEIT

DIVA
ARCHITECTURAL PERSPECTIVES ON INFORMATION VISUALIZATION

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit te Amsterdam,
op gezag van de rector magnificus
prof.dr. T. Sminia,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen / Wiskunde en Informatica
op dinsdag 8 mei 2001 om 15:45 uur
in het hoofdgebouw van de universiteit,
De Boelelaan 1105

door

Sebastianus Petrus Cornelis Schönhage

geboren te Haarlem

Promotor:	prof.dr. J.C. van Vliet
Copromotor:	dr. A. Eliëns

Preface

It was in 1981 when it all started. My dad bought an original IBM Personal Computer with a 4.77 MHz Intel 8088, 16 KB memory, two floppy drives and a black/green monochrome monitor. At that time, I was an 8 year old, curious little boy. Sunday mornings, while my parents were still asleep, I went down to the room where the computer was, booted it, thoroughly read the original MS DOS manual and played with the PC until I more or less understood why it did what it did.

The Acorn BBC was my first own computer that my parents gave to me when I was 11 years old. At that moment, most people I knew owned a Commodore 64 or an MSX computer and were playing games all the time. Since I only had a few and mostly boring games, I used my computer differently. I taught myself how to program the BBC in Basic and in Assembly.

My interests for computer graphics originate from about 1986 when *Sinterklaas* gave me an Amiga 500. Finally, I could play my games. However, programming still interested me very much. It was on the Amiga that I created my first *Visualization*. It was on a beautiful Sunday when Martijn and I had created a small Basic program that would draw a Mandelbrot fractal. I remember it like it was yesterday. After we had tested the program and were pretty sure it would work, we started the application. Since my Amiga was not that fast and we did not optimize anything at all to speed up the fractal calculation algorithm, the calculations took more than a couple of hours. We went to the beach with my family and when we came back it was there, in 65.536 colors: our first *apple man*.

Computer graphics, raytracing, 3D animation and so forth have always interested me. Just like computer programming has always remained an attractive way of spending time. Therefore, I was very happy that after my studies of

computer science I was given the opportunity to do research on visualization in the software engineering department of the Vrije Universiteit. The result of working on something that once started as my Sunday morning hobby is in your hands right now. I hope that you like reading it as much as I liked working on it for the past four years.

Acknowledgements

Without the help and support of many people this work would never have been finished. In particular I am very grateful to my mother, father and sister for their unconditional support!

Another person who I want to thank is Martijn who is probably the human being that I have seen most in my life. For the last 14 years we have seen each other almost each day, all day long. Thanks Martijn, I am going to miss you.

I want to thank all the people who were directly involved with the DIVA project in one way or another, namely Anton, Hans, Peter Paul, Ard, Edwin, Alex and Sana. Without your help, DIVA would never have been the way it is now. Additionally, I am grateful to everybody who read (parts of) this thesis and provided me with useful comments. In particular thanks to Anton, Hans, Martijn, Inge, Joris, Nico, Ard, and Frank.

During the last four year, I had a lot of nice colleages and good friends at both the Vrije Universiteit and ASZ Research and Development. Although I want to thank all of them, I want to mention explicitly Jacco, Nico, Jaap, Frank N, Frank C, Arno, Gerco, Thiel, Irmen, Job, Ard, Joris, Arne and Frederike.

Amsterdam,
October 2000

Bastiaan Schönhage

(bastiaan@schonhage.com)

More Information

More technical information and the source code of the DIVA prototypes can be found at:

http://diva.schonhage.com

Contents

CHAPTER 1

Introduction

In the Beginning ...
Was the Command Line.
Neil Stephenson.

A picture says more than 1,000 words is a well-known expression. And although it is not literally true, for then I would have drawn this thesis in about 80 pictures, it neatly expresses the power of visual information.

However, as always, good things never come alone. A considerable part of visual information is non-informative or even deceiving. And as such, the art of creating good visualizations is gaining interest in multiple research areas. In the field of computer science, the research focus is on supporting people in creating and presenting visualizations through computer-based systems.

The work described in this thesis looks at visualization mostly from a software engineering perspective. An approach comprising the search for requirements, high-level design and experiments through the creation of working prototypes forms the foundation of our research. However, creativity was not absent. Design, independent of whether that concerns software architectures or visualizations, requires both a thorough domain knowledge and a certain portion of inspiration.

1.1 Problems

If all problems concerning computer-based visualization had been solved and we were living the future of visualization (Chapter 2011), this thesis would not have been written. Or it would have had a different subject. However, from where you are now still more than 200 pages are left about computer-support for the process of visualization. And they aim at providing solutions to the following three major problems in information visualization.

1.1.1 Lack of multi-user support

Information visualization presents data in a graphical format. As we will discuss later, visual information is especially useful to **understand** information and to **communicate** information. Visualization software is mainly aimed at helping its users in understanding the data and discovering relationships within the data. For example, management information visualization tools support managers in understanding business processes and monitoring the business results of their organization.

Currently, visualization software is, however, mainly targeted at single users and most tools lack support for multiple users to work together. This is strange. Especially when visualization informs and supports people who have to make decisions. Collaboratively foraging the information can have a positive effect on the common understanding and consequently improve the quality of the decision.

Exchanging or communicating visualizations is currently not supported by most visualization tools. And when it is supported, it is usually limited to exchanging the resulting pictures instead of sharing the full visualization environment. Summarizing, our first problem is that although information visualization is often a multi-user activity, computer-based support is limited to single users.

1.1.2 Tight coupling

Visualization is becoming more and more common in today's tools. In addition to showing data as it is, tools allow for a visualization of the data. For example, with each new release of Microsoft Office, the visualization capabilities of Excel and Access have been extended.

The visualization capabilities are, however, tightly coupled with the data producing or managing application. They cannot be used independently of their environment. For example, a new simulation tool may contain extensive means of visualizing the (intermediate) results of a simulation. However, that

integrated visualization cannot be used to represent information from a different data source. Comparing the simulation visualization with a visualization of stored data is therefore cumbersome.

Separate visualization environments, on the other hand, often contain filters to import all kinds of data. Unfortunately, the link with the information provider is in that case a no-coupling-at-all link, as apposed to the tight-coupling in the case of built-in visualization. A separate visualization tool can import the results of the simulation but is incapable of showing intermediate data as the built-in visualization can. Summarizing, most present-day visualization solutions are either too tightly coupled or too separate.

1.1.3 Limited interaction

Interaction is an important feature of software. For example, the success of the direct manipulation interface is to a large extent due to its inherent support for straight-forward and direct interaction. But also in the realm of visualization, interaction is becoming increasingly important. *The best visualizations are not static images to be printed in books, but fluid, dynamic artifacts that respond to the need for a different view or more detailed information* (Ware 2000).

Most computer-supported visualization products claim to be interactive environments. The amount of interactivity, however, is mostly limited to the visual model. This means that the user can manipulate the resulting images of the visualization process but that interaction with the information source or the information producer is absent or very restricted. To increase the user's understanding of relations within the information, we advocate full interaction with every aspect of the visualization process.

1.2 The Diva Project

My Master's study of Computer Science at the Vrije Universiteit in Amsterdam resulted in an article entitled "Animating the Web" (Eliëns, van Ossenbruggen & Schönhage 1997). The paper describes how multiple media types and interactive contents could be integrated on the Web through a structured markup language (SGML). **Integration**, which was the keyword of that research, initiated a new research project focusing on the integration of simulation, animation and visualization in a hypermedia environment.

The project, which soon received the name DIVA, was part of a larger research project called *Simulation In Normative Systems*. Our contribution to this project is a software architecture to present the results of the simulation. However, the goal of the DIVA project has broadened from merely Web-presentation of simulations to interactive and collaborative visualization of information.

The goal of the Distributed Visualization Architecture (DIVA) project is:

> *Development of a distributed software architecture for interactive visualization of dynamic information. To support multiple users with different information requirements, the architecture must support multiple perspectives/views on the information. Additionally, simulations will be included as dynamic sources of data and information. Furthermore, to improve the usability of visualization, user-interaction and user-collaboration has to be incorporated into the architecture.*
>
> *To illustrate the architecture, prototype implementations will be built that allow multiple users to visualize data coming from shared information sources and simulations.*

Information visualization has received a lot of attention in recent years. However, most solutions are still immature in the sense that they are restricted to a limited set of information or to a limited ability of experimenting with the data and visualization. The main contribution of the DIVA project is to investigate the process of visualization and develop a flexible architecture that supports that process adequately.

A non-trivial part of the project consisted of investigating technical issues. Because of the rapidly improving techniques underlying distributed computing and 3D visualization (computer graphics), keeping up with state-of-the-art technology is time-consuming. However, in my opinion, a research project about distributed software architectures with only a marginal focus on technical issues is not very valuable. By studying technology and creating prototypes a much better insight into the material can be gained.

1.3 Perspectives on Visualization

Perspectives play an important role in the DIVA project. On a coarse level, we look at the concept of visualization from both an academic and a business perspective. On a more fine-grained level, visualization perspectives are inevitable to support multiple users in a visualization environment.

1.3.1 Visualization and architecture

Information Visualization and Software Architecture are the main subjects in this work. Although the two topics come from different disciplines, they are merged here in the sense that we investigate software architecture to support different aspects of information visualization.

Visualization and architecture also provide the two foundations on which this thesis is built. Some chapters mainly concern visualization aspects such as the

added value of interactive visualization or the theory and practice of visualization models. In contrast, other chapters focus on (software) architecture and use visualization only as the 'accidental' domain.

Hopefully, looking at the problem domain from multiple perspectives increases the understanding of the domain. A better understanding, subsequently, enables the development of better solutions.

1.3.2 Academic and business perspectives

An important influence on this research is the fact that it is a combination of research in an academic and business context. This has two major advantages. On the one hand, we have the academic freedom to investigate new and unproven topics. On the other hand, we have easy access to the 'real' world of users with 'real' problems. This has definitely contributed to the usefulness of the project by lifting it above the academic playground.

In addition to providing two different mindsets to look at the domain, both university and industry also have two distinct resources of people. Students from the Vrije Universiteit were a valuable addition to the project. Most of them were extremely productive and a large part of the prototypes is due to the effort of those students. On the other side, business managers and business engineers at ASZ and Gak made it possible to do case studies and to experiment with visualizations in a business context.

1.3.3 Visualization perspectives

So many users, so many perspectives is the title of an article that we published on the topic of collaborative visualization (Schönhage, Bakker & Eliëns 1998). It denotes the observation that although people are using the same information sources, they require different ways to view the information depending on their background and information requirements. Visualization perspectives are a means to look at information.

As we see it, perspectives play an important role in visualization. A single visualization of data is never enough. Multiple users, having multiple goals with the data require multiple perspectives on the data. Support for multiple perspectives is a topic that is still underexposed by most visualization tools.

1.3.4 The software engineering perspective

Software engineering can be seen as a combination of perspectives on the development of software comprising analytical, architectural, implementation, and organizational perspectives. In this thesis, we try to cover most of these perspectives. Therefore, we will discuss issues ranging from requirements to

implementation, and from architectural styles to organizational forces of introducing 3D visualizations. This range of issues might look like a farrago at first. However, in the end all perspectives contribute to the total picture of a powerful architecture that enables interactive and multi-user information visualization.

1.4 Structure of the Thesis

The structure of this thesis is as follows. Chapter 2, entitled *Information Visualization*, introduces the domain of this project. It illustrates how visual information can induce correct or incorrect decision making. Additionally, it introduces two types of visualization: scientific and information visualization.

The next chapter, *Management through Vision: a case study*, describes how business visualization can help managers in their process of decision making. It explains how effective visualizations can better inform a manager and thus improve the quality of the decisions. The evaluation of the case studies resulted in issues that are used as a source of input for the requirements of the DIVA architecture.

Chapter 4, entitled *Visualization Models: theory and practice*, covers related work, both in the theoretical and practical field. The theoretical part describes visualization models which describe the process of transforming data into visual representations. The second part of the chapter discusses practical visualization environments. These projects and (commercial) products are discussed according to their contribution to the process of creating visualizations.

In Chapter 5, *Diva: Distributed Visualization Architecture*, the core of this project is described: the DIVA architecture. According to three perspectives (conceptual architecture, software architecture, and information architecture) a discussion of how requirements are covered by the architecture is provided. To keep things manageable, the architecture is described in two phases. First, the chapter introduces the basic architecture covering the basic requirements. After that, we show how extra requirements are being incorporated by extensions to the basic architecture.

Chapter 6, *Experiments with Visualization and Collaboration*, provides a historical overview of the prototypes that have been created during the DIVA project. It illustrates how the software architecture can be instantiated in working systems and how technology has influenced the architectural choices. The prototype that experiments with collaborative visualization is described in somewhat more detail.

In the next chapter, entitled *3D Gadgets for Business Process Visualization*, the case study that was already described in Chapter 3 is extended to a three-dimensional visualization. It describes a collection of visualization primitives that can be reused for visualizations in a business context. Additionally, it

compares the 2D and 3D prototype according to usefulness, effectiveness and efficiency.

After a digression about experiments and extensions of DIVA in the previous two chapters, Chapter 8 describes an essential element of the DIVA architecture: *the Shared Concept Space*. The Shared Concept Space decouples information provider and visualizer through shared concepts. Additionally, it supports derived concepts which provide new information based on source data and derivation knowledge.

Chapter 9, *Distributed Objects: from Features to Styles*, goes back to a more abstract level. After all the details of distributed software architectures, 3D visualization models and shared data systems, this chapter tries to extract more general aspects of distributed object-oriented software. It discusses four architectural styles that represent four different means of connecting distributed objects and making them collaborate.

The purpose of Chapter 10 is to wrap up everything that has been said in the thesis. It summarizes the contributions and open issues of the DIVA project in the realms of Information Visualization and Software Engineering.

The last chapter (the *odd* chapter) must be seen as a gift. The chapter, entitled *The Future of Visualization*, describes a visionary scenario of what will be possible when people in an organization are connected by a collaborative, distributed visualization system through which they are able to exchange thoughts, arguments and other types of information.

1.4.1 Visualization of the structure

Since this dissertation concerns information visualization, Figure 1.1 contains a visualization of the structure of the thesis. It represents the relations and important dependencies between the chapters and parts comprising the thesis.

On a coarse level, the thesis consists of six parts: *introduction, visualization theory & practice,* DIVA *architecture, distributed software architectures, conclusion and future*. The parts are represented in Figure 1.1 by the large boxes. All parts consist of one or multiple chapters represented by the small numbered boxes.

The large block arrows between the parts represent important dependencies. They indicate that particular topics from the previous part are necessary to fully understand the discussion in the next part. The required knowledge is given in the dashed box near the arrow. For example, the architectural discussion given in Chapter 5 builds upon visualization issues and requirements described in the context of our own case studies (Chapter 3) and related work (Chapter 4).

Dependencies between individual chapter within the blocks are illustrated by the small arrows. For example in the DIVA architecture part, it is necessary to read the theoretical chapter (Chapter 5) before reading any of the chapters discussing the practical deployment of the architecture.

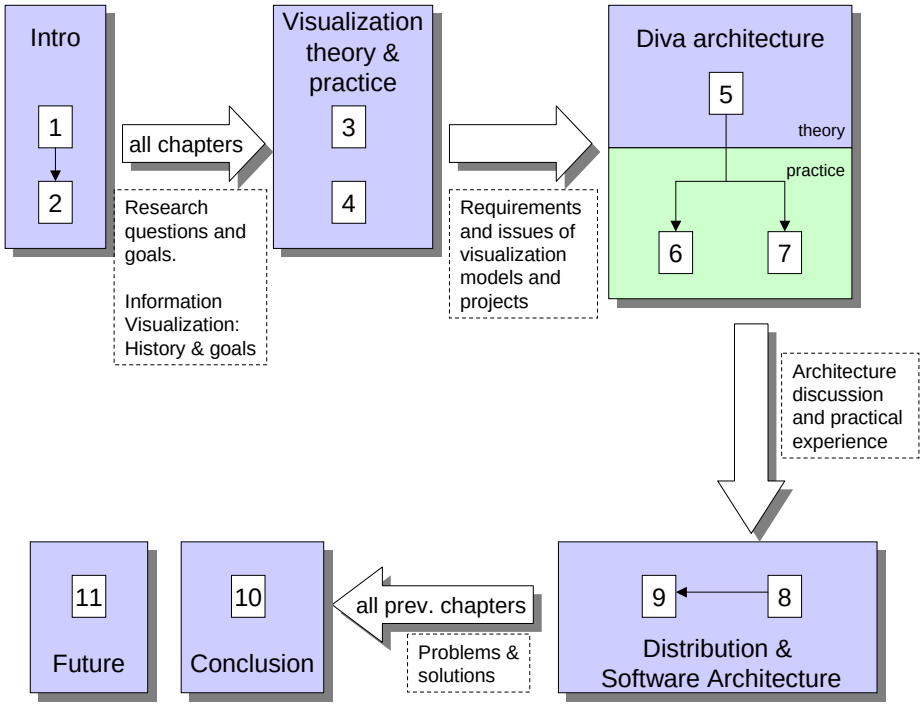


FIGURE 1.1: Visualization of the structure of this thesis that illustrates the dependencies between the chapters

1.4.2 Paths through the dissertation

Of course everybody should read the full text of the dissertation. However, it is possible to indicate at least two paths through the book, which skip some chapters or entire parts of the text. The first path is useful to people who are mainly interested in (information) visualization and do not care much about software architecture. The second path is for people interested in software architectures. People who want to follow the visualization path should mainly concentrate on Chapters 1, 2, 3, 4, (5), (6), 7, 10, and 11 where the chapters between brackets are somewhat less important. For those more interested in the software architecture path, please concentrate on Chapters 1, (3), 5, 6, (7), 8, 9 and 10.

1.5 Publications

A lot of the work presented in this thesis has been published elsewhere. The next overview describes the publications that this dissertation is based on.

The case study described in Chapter 3 was presented at the *Information Visualization 2000 (IV2000)* conference in London (Schönhage & Eliëns 2000*b*).

The DIVA architecture (Chapter 5) is present in most publications. However, the core architecture is described in (Schönhage & Eliëns 1997) and (Schönhage & Eliëns 1998). These papers were presented at *New Paradigms in Information Visualization and Manipulation (NPIV '97)*, and at the *International Conference on Digital Convergence: the Future of the Internet and WWW*, respectively. The latter paper has also been published as a book chapter in (Earnshaw & Vince 1999).

Collaborative visualization in DIVA, as covered in Chapter 6, was previously presented at the *IFIP 13.2 Working Conference on Designing Effective and Usable Multimedia Systems* in Stuttgart, Germany (Schönhage et al. 1998).

Two publications underlying this thesis were presented at the *International Conference on the Virtual Reality Modeling Language and Web 3D Technologies*. The first one in 1999 (Schönhage & Eliëns 1999*a*) is part of Chapter 6. The second paper (Schönhage, Eliëns & van Ballegooij 2000), which covers 3D gadgets for business process visualization, forms the basis for Chapter 7.

The Shared Concept Space is discussed in Chapter 8. Previously, the material was presented at *Distributed Objects and Applications (DOA'00)* in Antwerp (Schönhage & Eliëns 2000*a*).

Finally, the material discussed in Chapter 9 is based on two workshop presentations. The first one (Schönhage & Eliëns 1999*c*) was presented at *Engineering Distributed Objects (EDO99)*, which was part of the *International Conference on Software Engineering (ICSE1999)*. The second paper (Schönhage & Eliëns 1999*b*) was part of the *Nordic Workshop on Software Architecture (NOSA '99)*.

CHAPTER 2

Information Visualization

As for a picture, if it isn't worth a thousand words, the hell with it.
Ad Reinhardt

Computer graphics and visualization offer "intelligence amplification" (IA) as compared to artificial intelligence (AI).
Fred Brooks

Information is an essential source for good decision making. Without a city plan, people will get lost in an unknown city; without clocks and time schedules, you never know when a train will arrive; and without marketing information, deciding whether or not to market the new product remains a high-risk guess. The list of examples is endless and because information is so highly integrated in our lives it does not strike us anymore that it is so essential. It is the **absence** of good or enough information that is notable.

Information comes in a plentitude of appearances. It can be a table of numbers, a list of rules, a book of laws, a collection of photos or a structured drawing of underground connections. For our purposes, we will distinguish between visual and non-visual information. **Visual information** comprises information of which a major part consists of visual elements such as drawings, photos and graphs. **Non-visual information** mainly consists of verbal elements in the form of text and numbers.

Structure The subject of this chapter is *Information Visualization*, which is essentially the process of transforming non-visual data to visual information. To understand why this adds value to understanding information, we must first take a look at visual information itself. When we appreciate the usage of visual information and know the merits of good information design, we will take a look at attempts to create visualizations in a scientific context (Section 2.2). A consecutive development is the visualization of more abstract data. This field of study comprises visualizations of large hierarchical and complex data structures as we find them in business information or the internet. Information visualization, as this discipline is called, is covered in Section 2.3. The usefulness of visualizations to support users can be increased by allowing them to interact with the visual information. Section 2.4, therefore, gives a quick overview of some interaction techniques. Finally, to illustrate the usage of information visualization, some typical application areas are discussed in Section 2.5.

2.1 Visual Information

Before investigating the concept of visual information in more detail, let us first look at information itself. According to Merrian-Webster's dictionary[1] information is *1) the communication or reception of knowledge or intelligence* and *2) knowledge obtained from investigation, study or instruction*. While thinking over the definition, two observations are quickly made. The first observation is that information is always aimed at increasing *knowledge* (otherwise it would just be factual data). In other words, information is only useful when it satisfies some information needs. A second observation is that information like knowledge is amorphous, i.e. it has no form by itself but has to be instantiated to be communicated. Visual information is one means to present information.

With visual presentation being one means to show information, what are the other options? Actually, a vast array of information presentation forms exist. Text or speech is one example. A large part of information transfer between people occurs by means of speech (conversation, radio) and prose (newspapers, books). More examples of information presentations are numbers, formulas and other symbol-oriented techniques.

Each type of information presentation has its unique properties and, hence, is suitable to be applied in different kinds of communication. For example, speech is appropriate to quickly exchange simple information between people. However, to communicate the development of prices on the stock-market a visual presentation is better suited. Why? Mainly because visual information gives quick insight using the high bandwidth of human's perceptual abilities.

An essential characteristic of visual information in contrast to, for example, textual information is the fact that it is non-linear. A text has an intrinsic order in which the information is poured whereas visual information is mostly

[1] www.m-w.com

random access. Viewers glance at a picture to get an overview and then look at particular parts of the image in more detail according to their interests at that moment. The choice of deciding what is relevant has shifted from information producer to consumer.

Bertin (1977/1981) characterizes the usefulness of visual information by describing three levels of reading a diagram: read fact, read comparison and read pattern. **Read fact** looks at a single number or another single data element. **Read comparison** concerns two or more parts of the diagram and compares them according to position, size, color, etcetera. **Read pattern** takes the complete diagram into account and searches for large-scale patterns.

Visual information is ubiquitous in our daily lives. For example, the weather report comes up with little clouds when rain is expected whereas a big sun forecasts nice beach weather. Every day, newspapers make use of graphs to show exchange rate fluctuations of the dollar and euro. In addition, advertisements that are trying to convince you to use their brand of toothpaste, show animations of decaying teeth that were not brushed properly. We could go on for a while, but the illustrated point is clear: visual information has the potential to communicate information quickly independent of whether the goal is to inform or to convince.

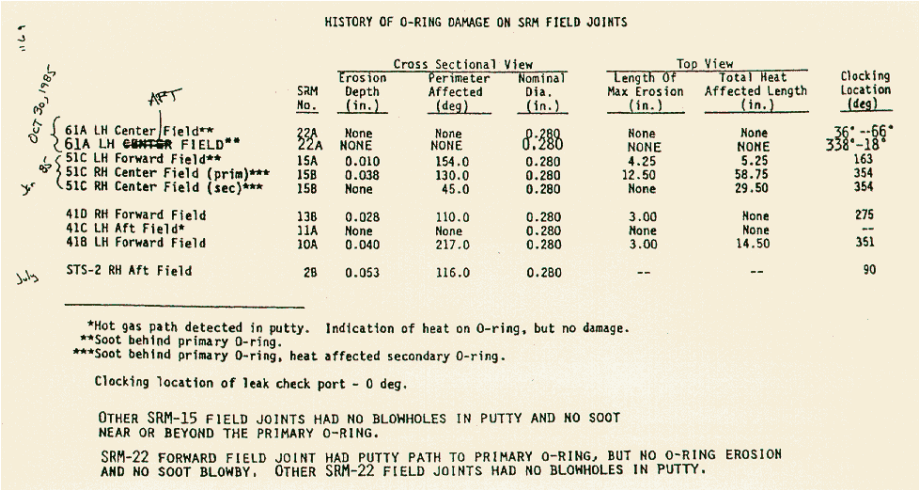
2.1.1 Information design

Designing visual information that is useful and effective is not straightforward. Tufte's trilogy on (visual) information design (Tufte 1983, Tufte 1990, Tufte 1997) abounds with examples of flaws. A compelling example is the fatal decision to launch the space shuttle Challenger on January 28, 1986. The shuttle exploded because two rubber O-rings leaked while they lost their resiliency due to the low temperature. The day before the launch, however, the engineers who designed the rocket faxed NASA thirteen charts indicating that the launch should be postponed. Unfortunately, the charts were unconvincing and unclear with respect to the relation between temperature and O-ring failure.

The night before the launch, the NASA officials discussed the problem with the rocket-designers' managers. And although this was the engineers' only no-launch recommendation in 12 years, the NASA decided that the evidence was inconclusive. Next morning, the Challenger exploded after 73 seconds.

In his discussion of the Challenger accident, Tufte (1997) states that the cause of the wrong decision of launching the space shuttle comes down to the inability to assess the link between low temperature and O-ring damage on earlier flights.

One of the charts, reprinted in Figure 2.1, shows the history of eroded O-rings on previous launches. The leftmost piece of text describes the damaged ring while the first column contains the launch number. The other columns indicate the measured damage of that particular ring.



HISTORY OF O-RING DAMAGE ON SRM FIELD JOINTS

	SRM No.	Cross Sectional View: Erosion Depth (in.)	Cross Sectional View: Perimeter Affected (deg)	Cross Sectional View: Nominal Dia. (in.)	Top View: Length Of Max Erosion (in.)	Top View: Total Heat Affected Length (in.)	Clocking Location (deg)
61A LH Center Field**	22A	None	None	0.280	None	None	36°--66°
61A LH ~~CENTER~~ FIELD**	22A	NONE	NONE	0.280	NONE	NONE	338°-18°
51C LH Forward Field**	15A	0.010	154.0	0.280	4.25	5.25	163
51C RH Center Field (prim)***	15B	0.038	130.0	0.280	12.50	58.75	354
51C RH Center Field (sec)***	15B	None	45.0	0.280	None	29.50	354
41D RH Forward Field	13B	0.028	110.0	0.280	3.00	None	275
41C LH Aft Field*	11A	None	None	0.280	None	None	--
41B LH Forward Field	10A	0.040	217.0	0.280	3.00	14.50	351
STS-2 RH Aft Field	2B	0.053	116.0	0.280	--	--	90

*Hot gas path detected in putty. Indication of heat on O-ring, but no damage.
**Soot behind primary O-ring.
***Soot behind primary O-ring, heat affected secondary O-ring.

Clocking location of leak check port - 0 deg.

OTHER SRM-15 FIELD JOINTS HAD NO BLOWHOLES IN PUTTY AND NO SOOT NEAR OR BEYOND THE PRIMARY O-RING.

SRM-22 FORWARD FIELD JOINT HAD PUTTY PATH TO PRIMARY O-RING, BUT NO O-RING EROSION AND NO SOOT BLOWBY. OTHER SRM-22 FIELD JOINTS HAD NO BLOWHOLES IN PUTTY.

FIGURE 2.1: The original chart only illustrates the history of O-ring damage, without making any relation to the temperature. Courtesy of Graphics Press.

Tufte (1997) discusses that the chart has some serious flaws. A first problem is that O-ring damage is described by six types of damage in the right-side columns of the chart. Thus comparing the severity of the damage between two individual cases is cumbersome. To be able to compare the different launches, only a single index for damage is necessary, e.g. the weighted sum of the individual numbers. A second essential problem is the fact that the chart does not provide data about the assumed cause of the failure, temperature. Therefore linking the cause and effect of the damage is impossible.

To investigate the Challenger accident later, a presidential commission was presented some more charts. Unfortunately, several of these charts still did not get it right. Figure 2.2 is one such chart. Each rocket in the figure represents a successful launch whereas marked areas in the small rockets indicate damages to the primary or secondary O-ring.

According to Tufte (1997), the fatal flaw in this chart is that the time-based order of elements is completely wrong. The sequential order illustrates the issue whether there is a *time trend* in O-ring damage. However, this is not what we are looking for. To answer the question whether there exists a *temperature trend* in O-ring damage imposes an ordering of the rockets by temperature instead of by time.

Other problems with the chart as pointed out in (Tufte 1997) are:

- **disappearing legend** — the legend has been presented in previous charts but is not available in this chart

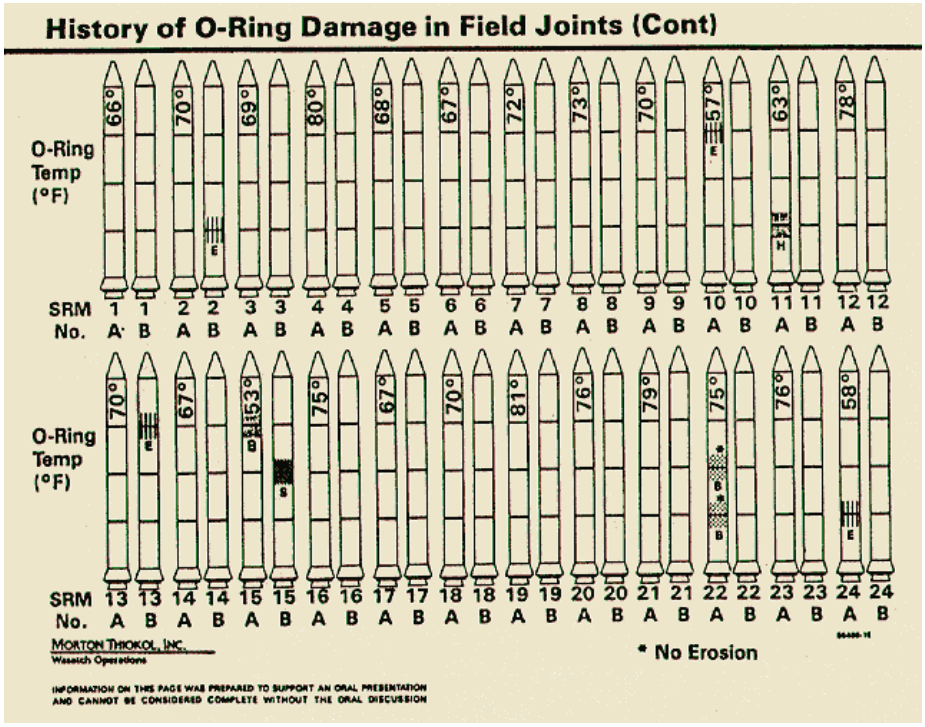


FIGURE 2.2: This chart, presented during an investigation of the accident, arranges damage according to time and is therefore not capable of showing the temperate trend. Courtesy of Graphics Press.

- **chartjunk** — the strongest visual presence in this graph is generated by the outlines of the rockets instead of by important underlying data
- **lack of clarity in depicting cause and effect** — again the O-ring damage is depicted in several places (instead of a single index) and relating it to the temperature is cumbersome.

Tufte has created a new scatterplot (Figure 2.3) that takes all criticism into account and clearly displays the relation between temperature and failure. The graph clearly illustrates the serious risks of a cold weather launch. Actually, the graph shows that every launch below 66°F had problems. Additionally, the risky extrapolation beyond all previous experience is expressed convincingly by the huge gap between known data points and the discussed area.

The Challenger crashed, not because the engineers were unaware but, because they were unable to adequately depict their thoughts and consequently convince the NASA officials to postpone the launch. This and many other examples suggest the following message: *there are right and wrong ways to show data; there are displays that reveal the truth and displays that do not* (Tufte 1997, p.45).

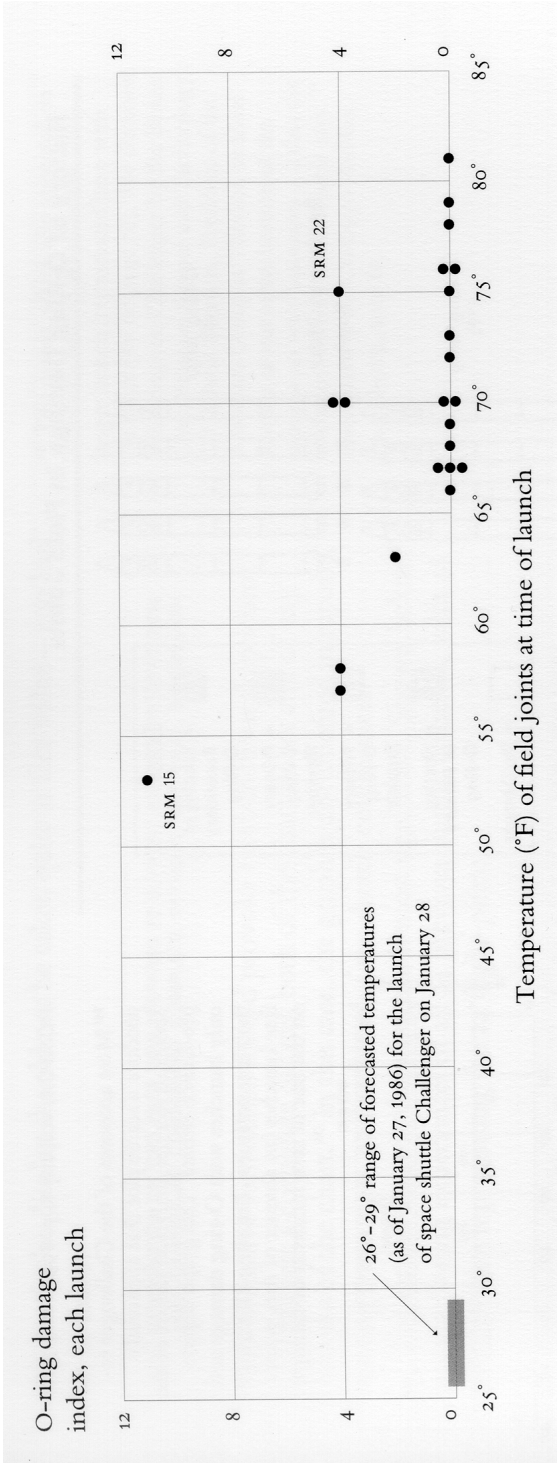


FIGURE 2.3: Tufte's scatterplot clearly depicts the relation between temperature and failure. Courtesy of Graphics Press

2.2 Scientific Visualization

In the November 1987 issue of *Computer Graphics*, McCormick, DeFanti & Brown (1987) introduced the concept of visualization into the scientific domain. They presented visualization as a means to observe information. By presenting complex data visually, scientists could benefit from a better understanding of the data under study.

In previous years, computer technology had offered scientists in the fields of physics, chemistry and meteorology the power to simulate complex models. Using these simulations, researchers were able to verify theories by computing results and comparing them with measured values. Additionally, simulation enabled *virtual experimentation*. A problem with simulations of increasing complexity is that resulting data also becomes harder to interpret. Presenting results of simulations and computations visually might help to better understand the results.

> **Visualization** is a method of computing. It transforms the symbolic into the geometric, enabling researchers to observe their simulation and computations (McCormick et al. 1987).

The definition given above makes clear that visualization is a method to transform non-visual data to visual information. The purpose of this transformation is to increase insight into the complex material of scientific models and their simulations.

The criteria discussed in the previous section about visual information and information design also apply to (scientific) visualization. In short, there are visualizations that reveal the useful information and visualizations that suggest wrong conclusions. Therefore, it is not enough to just apply some visualization technique to a dataset. In contrast, visualizations have to be carefully designed. Depending on the context, particular visualization techniques are better suited to reveal relationships in data than others.

2.2.1 Scientific visualization techniques

Source data that is used in scientific visualization often has some important characteristics. Shneiderman (1996) discriminates seven data types in his **data type by task taxonomy** which characterize the task-domain information objects and lead to distinct visualizations. The data types are: 1-D linear, 2-D map, 3-D world, temporal, multi-dimensional, tree and network.

Datasets, which are the source of scientific visualization, consist of an **organizing structure** and **data attributes** associated with the structure. In scientific visualization, the organizing structure often represents the physical space of

the data. For example a dataset about the outside temperature of several locations in a country has a two-dimensional organizing structure that reflects the shape of the country.

Data attributes can be assigned to components of the structure, i.e. points, edges or faces. The data attributes themselves can be of arbitrary dimension. Examples are scalars (single value) and vectors (magnitude and direction).

The next paragraphs will introduce a few important techniques to visualize datasets. The set of techniques comprises color mapping, isosurfaces and particle animation. More scientific visualization techniques can be found in, amongst others, (Schroeder, Martin & Lorensen 1996).

Color mapping

Color mapping is a useful technique to visualize scalar data in a two dimensional grid or a 2D cut through a three-dimensional space. Scalar values located at (x, y) are mapped to a particular color independent of the value of x and y. The mapping can take place by means of a **lookup table** which contains color values for all, or ranges of, scalar data. A more general form of mapping scalar data to a color is achieved by a **transfer function**. To create a useful and effective visualization the choice of the lookup table or transfer function is essential and usually requires a certain amount of experimentation.

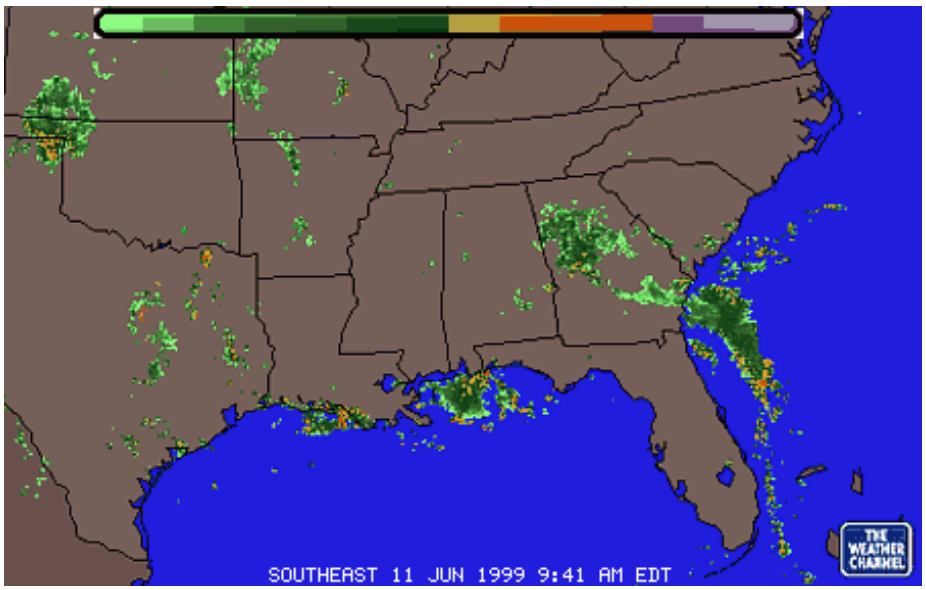


FIGURE 2.4: Color mapping precipitation on the map of the USA. Courtesy of the weather channel (www.weather.com).

As a example of color mapping take a look at Figure 2.4, which visualizes the precipitation on June 11, 1999 in the southeastern part of the USA. The data which is used as the source of this color mapping is one-dimensional, scalar data (the amount of rainfall) ordered on a two-dimensional grid (the surface of the USA). The resulting image presents the scalar data mapped to colors, ranging from green for light rain to red for heavy showers. To be able to interpret the resulting visualization, the color map is projected on top of a map of the corresponding part of the country.

Isosurface

Isosurfaces are the 3D version of a more general technique called *contouring*. In this technique we draw the boundaries (in 2D with lines, in 3D with surfaces) of constant scalar value to create regions in the dataset. This way, we can discover areas which have similar values and therefore a strong relationship.

As an example, look at Figures 2.5 and 2.6, both visualizing the same MRI-scan (Magnetic Resonance Imaging) using different isosurfaces. An MRI-scan measures the variation in a magnetic field in response to radiowave pulses. The dataset used has the structure of a 3D grid (actually a series of slice planes) with scalar values assigned to datapoints. By applying an isosurface visualization to this dataset, we can reveal the skeletal body of the scanned head (Figure 2.5) as well as the skin (Figure 2.6). We can show both images based on the same dataset because the magnetic resonance of bones and skins are fairly distinct.

Particle animation

The last technique discussed here visualizes vector data instead of scalar data. The example we will use visualizes wind speeds calculated using a simulation of a tornado. The topology of the dataset is a three-dimensional grid and the associated data values consist of 3D vectors denoting speed in x, y and z-direction.

A straight-forward way to visualize this dataset is to draw arrows representing the vector data at each point in the three-dimensional grid. A more dynamic visualization, however, uses particle animation to visualize the tornado. As a metaphor, think of throwing some leafs into a whirlwind and see them swirling around.

We create this animation by inserting particles in the 3D space and during each step of the animation we move each particle according to the vector at the current position of that particle. By playing the animation frames, we can see the particles traveling through the 3D space illustrating the streams in the dataset. Figure 2.7 contains a single frame of an animation in which the particles also leave a little trace behind to better show the direction in which they are moving.

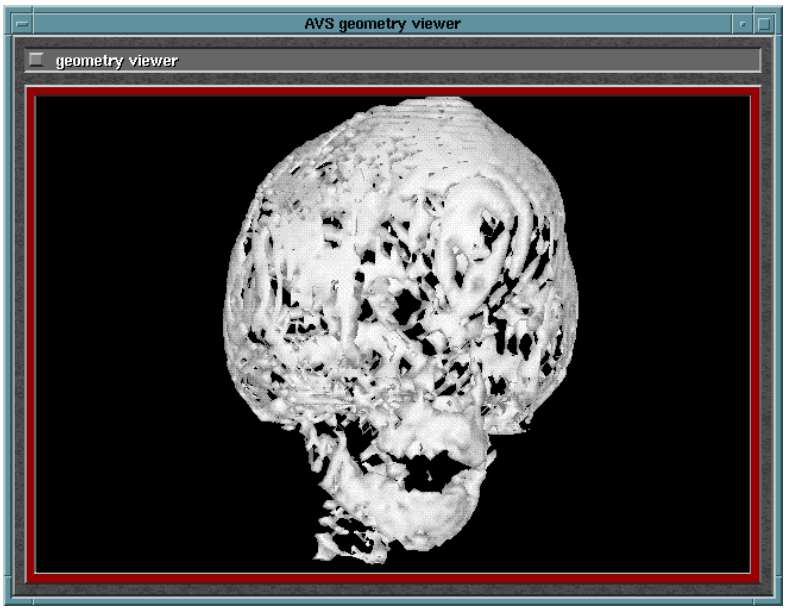


FIGURE 2.5: Isosurface revealing the bones of an MRI-scan.

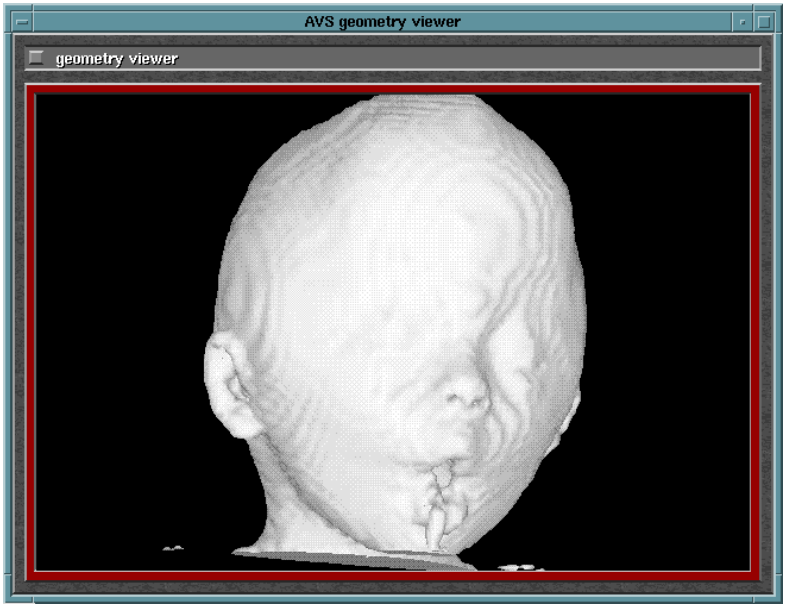


FIGURE 2.6: Isosurface showing the skin of the same MRI-scan using different parameters.

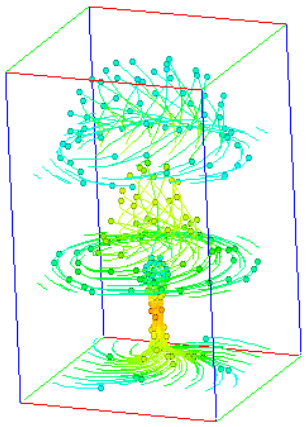


FIGURE 2.7: A particle animation of a tornado illustrates how elements would swirl through the air.

2.3 Information Visualization

During the last decade, a new visualization research focus has emerged, intended to visualize **abstract** information. Robertson, Card & Mackinlay (1993) state "Information visualization attempts to display structural relationships and context that would otherwise be more difficult to detect by individual requests." In other words, information visualization transforms abstract information into visual information enabling all the advantages of quick insight and pattern recognition that are inherent in visual information.

2.3.1 The purpose of information visualization

Scientific visualization usually borrows its representations from the physical world as can be seen from the MRI-scan and tornado examples. Because information visualization focusses on abstract data, mapping to the physical world is difficult or impossible. Hence, a key research issue of information visualization is "to discover new visual metaphors for representing information and to understand what analysis task they support" (Gershon, Eick & Wright 1997). Another important issue concerns the ease of use because information visualization has a more diverse audience in contrast with the specialized, technical users of scientific visualization. Table 2.1 compares information visualization with scientific visualization on four characteristics, audience, task, input and input quantity.

In their excellent introduction to *Readings in Information Visualization: using vision to think* Card, Mackinlay & Shneiderman (1999) compare visualization

TABLE 2.1: Information visualization compared with scientific visualization (Gershon, Eick & Wright 1997).

	Audience	Task	Input	Quantity
Scientific Visualization	Specialized, highly technical	Deep understanding of scientific phenomena	Physical data, measurements, simulation output	Small to massive
Information Visualization	Diverse, widespread, less technical	Searching, discovering relationships, including action (fast, many times!)	Relationships, nonphysical data, information	Small to massive

with **external cognition** where external representations are used to amplify the acquisition and use of knowledge. Their definition of information visualization, stated below, explicitly contains the purpose of visualization: *the purpose of visualization is insight, not pictures* (Card et al. 1999, p.6).

> **Information Visualization:** the use of computer-supported, interactive, visual representations of abstract data to amplify cognition (Card et al. 1999, p.6).

In (Schönhage et al. 1998), we have discussed that people use visual representations of information for two different purposes. First, visualization is often used to **understand information**. A visualization gives quick insight into information using humans' remarkable perceptual abilities (Shneiderman 1998). Second, visual representations are used to **communicate information** to other people. Shneiderman (1998, p.522) states that the bandwidth of information presentation is potentially higher in the visual domain than it is for media reaching any of the other senses. In the first case, when using visualization to understand information, we often apply it individually (although it is surely useful to try to understand information in a group process). In the latter case we are communicating with other people because we try to illustrate something, or we want to convince them of our point of view.

2.3.2 Expressive and effective visualization

Now that the purpose of visualization is clear, it is useful to look at how successful visualizations are at *amplifying cognition*. The success-ratio of visualization is covered by two criteria: expressiveness and effectiveness (Mackinlay 1986).

A mapping from data to a visual structure is **expressive** if all data and only the data is represented. So, all existing data is available and no new, unwanted data is introduced. A visualization is **effective** when it adequately supports the user in finding relationships or understanding the data. Hence, deciding

whether a mapping is effective can only be done by taking the information requirements of the perceiver into account.

Although proposals exist to automatically generate expressive and effective visualizations, the lion's share of visualizations are still made by hand. The tool support in the manual process of creating visualizations ranges from low-level graphics APIs leaving everything to the 'visualization programmer' to prefabricated visualizations in which only the data has to be poured. The higher-level the tool support is, the less complex but also the less flexible a user can create or adapt a visualization. As a consequence, high-level support is advisable for the occasional user whereas expert users who want to control every aspect of the visualization need a more flexible approach.

A lot of research is currently being done in the field of designing easy to use systems to quickly create powerful visualization tools for exploration without programming every bit and piece of it. Most of those systems provide a visual programming language to perform the mapping from the semantic domain to a visual presentation. For example, VANISH (Kazman & Carriere 1996*a*) and (Kazman & Carriere 1996*b*) is such a system where users can very quickly (within one hour) create or adapt visualizations in a visual manner.

A good example of an approach that covers both the low-level and higher levels of visualization creation is Visage (Derthick, Kolojejchick & Roth 1997) and (Kolojejchick, Roth & Lucas 1997). The system contains a number of high-level *information appliances* to navigate through and explore data. Additionally, it contains low-level tools to create new appliances. Visage combines different interaction styles and, thus, is appropriate for both novices and experts.

2.3.3 Examples

A simple, yet powerful, information visualization mechanism, which is often used, is the two-dimensional scatterplot. A **scatterplot** presents the structural relation between two dimensions, e.g. the relation between damage and outside temperature at space shuttle launches. The graph consists of two axes representing the dimensions and a collection of points representing the data elements. As an example, look at Figure 2.8 depicting body height versus weight for a certain population. In the picture, which is based on the *Statistics Glossary*[2], we can immediately see that a larger height usually comes with a larger weight, thus revealing a relation between body height and weight. As an extension to the standard scatterplot, the plots may depict additional data dimensions by size, shape or color of the points.

Datasets having different structures (dimensions) and attributes require alternative forms of visualization. Whereas a scatterplot is useful to present two-dimensional quantitative data, a **conetree** (Robertson, Mackinlay & Card 1991) is more suitable to visualize large hierarchical datasets. As examples of such

[2]http://www.stats.gla.ac.uk/steps/glossary/presenting_data.html

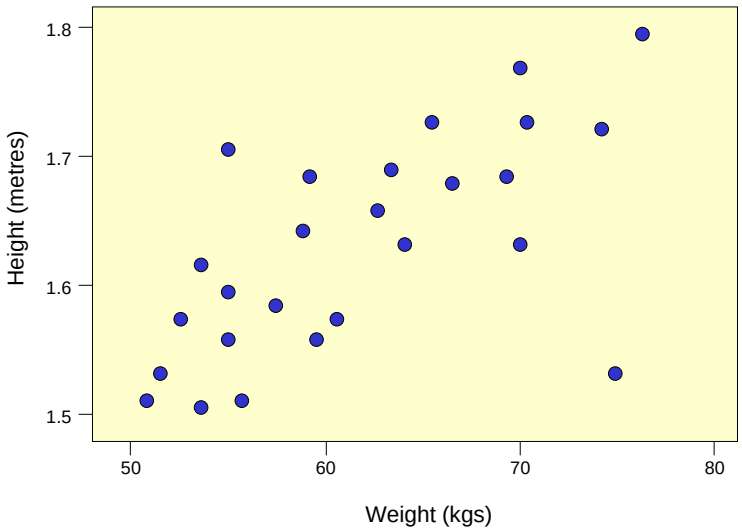


FIGURE 2.8: A scatterplot quickly relates possible relations between two dimensions, in this figure it shows that a larger height usually comes with a larger weight.

datasets, think of directory structures in large filesystems, or the organization of pages on a Web site. A conetree is the three-dimensional counterpart of a 2D tree. The children of a specific node are organized in a circle resulting in a cone that comprises parent node and children. Figure 2.9 contains a conetree that visualizes the directory structure of the files that constitute this dissertation.

The conetree in Figure 2.9 is not a static two-dimensional picture but an interactive 3D object around which we can navigate to view the tree from a different angle. The constellation of the tree itself can be altered by dragging the cones, for example to roll a particular part of the tree to the front. Another useful feature is achieved by shift-clicking a (leaf) node. The conetree automatically rotates its branches and cones in such a way that the selected path, in this case the full filename, is facing the user.

As a last example of information visualization, Figure 2.10 visualizes product sales in the USA. The sales quantity is mapped onto the height of the related state. This makes it relatively easy to compare the sales of different states with each other. Additionally, color is used to emphasize whether sales are doing good (green) or bad (red).

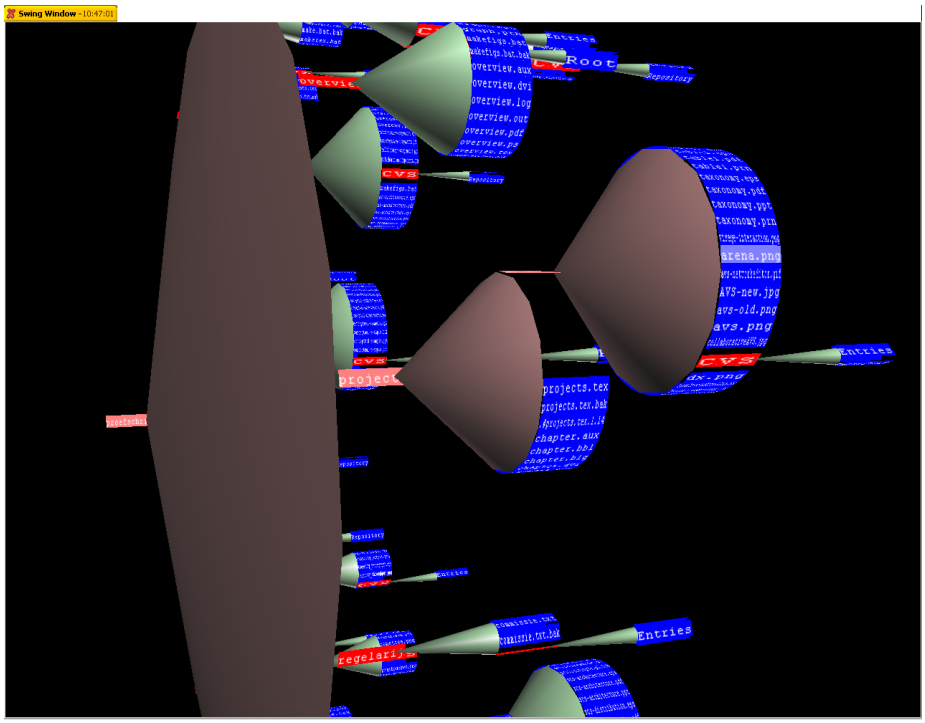


FIGURE 2.9: A conetree representing a directory structure

2.4 Interaction Techniques

The definition of information visualization given on page 22 includes the word 'interactive'. Interactivity means that users cannot only observe the visualization but moreover play with it. The visualization changes in response to the user's actions such as zooming in on details, navigating to other visualizations or altering the applied visualization techniques.

Many ways of interaction with a visualization exist. Specific visualizations often even require specific means of interaction. However, some interaction mechanisms are generic enough to be useful for most visualizations. We will now describe three basic interaction techniques which are intended to retrieve more or related information. However, each of the techniques achieves this in a somewhat different manner.

Zooming in on information is a powerful interaction technique. For example when more detailed information is required about the sales visualization of Figure 2.10. By clicking on a particular state we are able to **drill-down** to more detailed information of that state. The map of the United States is replaced by a map of the selected state which shows the number of items sold in important

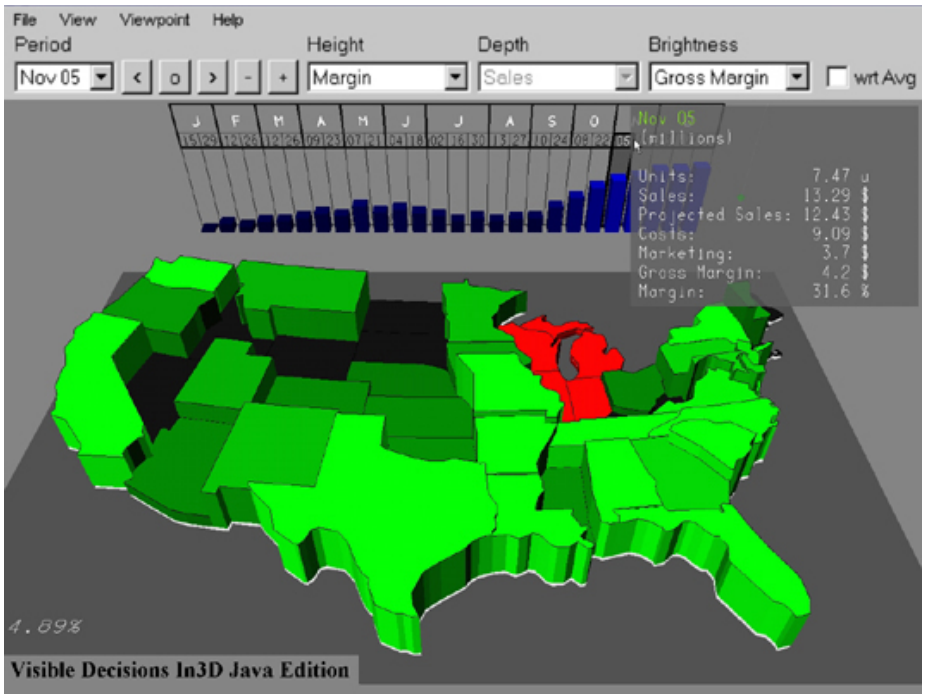


FIGURE 2.10: Information Visualization of sales in the USA. Courtesy of VDI (www.vdi.com).

cities. Furthermore, we can zoom in on a particular city or even on the number of items sold at a single location. Important here is that we use the *visualization* itself to navigate through the data.

Drill-down is only one technique to navigate and interpret data, other important techniques include brushing and hyperlinking. **Brushing** allows to retrieve more detailed information about parts of a visualization without changing the visualization itself. By moving a pointing device over a particular component in the visualization extra information appears on top of the selected object. An example of brushing is given in Figure 2.11. The advantage of brushing is that more detailed information can be retrieved quickly without replacing the current visualization. A simple mouse movement is enough to reveal, for example, the numbers on which the visualization is based.

Hyperlinking is a technique stemming from hypertext and the World Wide Web. By activating a hyperlink a new "document" is loaded, which can be a different visualization, a text document or an HTML-page. This new document can replace the current visualization but might also be displayed in a new window or frame. By using hyperlinks, visualizations can be integrated with other resources to provide extra context. For example, hyperlinks in a man-

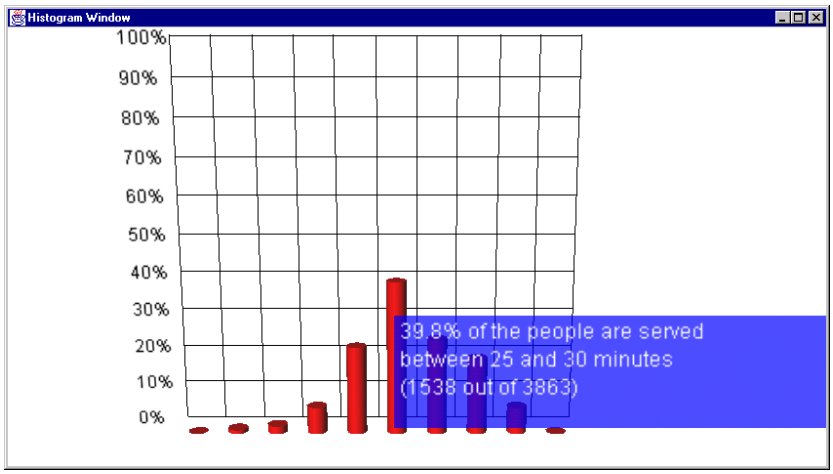


FIGURE 2.11: Brushing reveals extra information

agement information visualization can provide background information about the meaning of elements in the visualization or the business process currently visualized.

2.5 Visualization Application Areas

Since information visualization is capable of visualizing any kind of data, application areas thereof are relatively diverse. For example, in a medical setting a patient's syndromes can be visualized (Plaisant, Milash, Rose, Widoff & Shneiderman 1996). LifeLines, which is the name of the system, helps medical specialists to quickly see which diseases a patient has suffered from. As a completely different example, consider FilmFinder (Ahlberg & Shneiderman 1994) which assists home users in their search for interesting movies.

However, two user tasks for which visualization is deployed recur in the majority of application areas: information retrieval and decision support. Therefore, this section will look at those tasks in somewhat more detail.

2.5.1 Information retrieval— visual queries

Information visualization is very good at presenting large amounts of data and providing good overviews. However, users are often interested in particular data elements or relations within a small subset of the data. Therefore, a visualization has to provide means for (preferably dynamic) information retrieval.

Comparable to what SQL (Structured Query Language) is for databases, **visual queries** are a means to adapt the data source of visualizations. By pushing (radio) buttons and adjusting sliders, users can visually filter and retrieve information.

The visual approach of querying databases has some important advantages and disadvantages as described in (Shneiderman 1994). Among the advantages are the fact that it is a *rapid, safe and even playful method* (Shneiderman 1994), allowing a wider range of people to explore the correlations in the data. Disadvantages all come down to the poor match between the dynamic query approach with current hardware and software. For example, to access a standard database using dynamic visual queries, application-specific programming and data conversions are still necessary.

The FilmFinder example (Ahlberg & Shneiderman 1994) illustrates the use of information visualization and visual queries. The FilmFinder system is a tool to explore a film database visually. The system combines a starfield display to visualize results, sliders (dynamic queries) to browse through the information and a tight coupling of the interface components to increase the usability of the system.

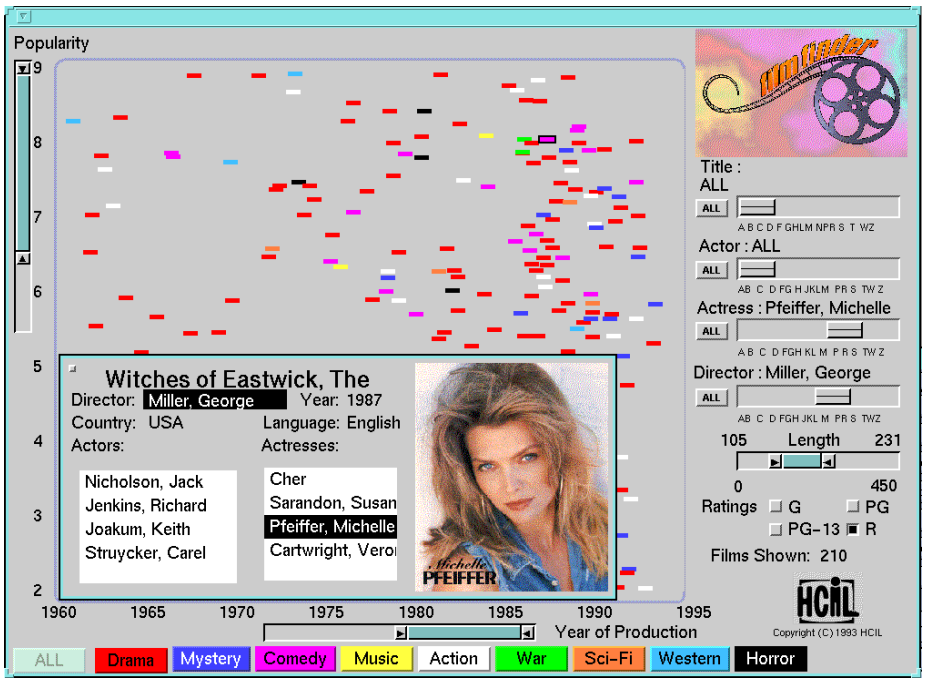


FIGURE 2.12: FilmFinder allows users to visually browse through a movie database using dynamic visual queries. Courtesy of University of Maryland.

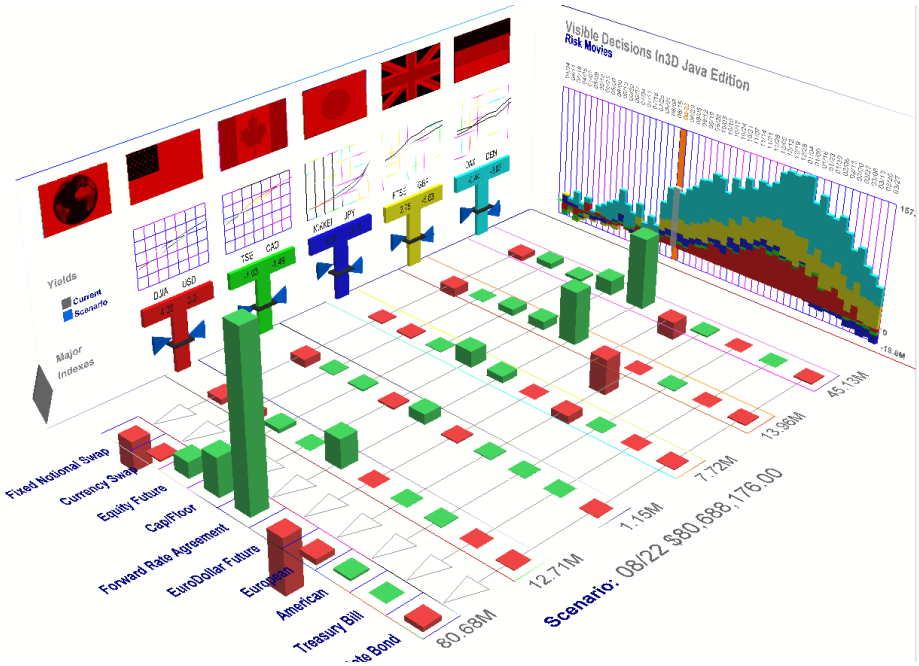


FIGURE 2.13: This 3D business visualization summarizes a 60 page report. Courtesy of VDI.

The query result is continuously represented in a starfield, see Figure 2.12. The x-axis represents time whereas the y-axis represents a measure of popularity. The color of the spots in the starfield represents the genre of the film. Using the sliders on the right-hand side of the figure, users can look for a specific title, actor, director, etcetera. Because of the tight coupling, a movement of the slider immediately results in an updated starfield. By right-clicking on a specific dot, more information about a particular movie can be obtained. In Figure 2.12 this is done for *the Witches of Eastwick*.

2.5.2 Business visualization— decision support

Only recently, information visualization is starting to enter the business world to give form to visual decision support. Benefitting from all the advantages of visual information such as quick insight and details on demand, decision makers have better information available to manage their businesses. For example, Wright (1995) proposes a combination of three-dimensional visualizations and animation to inform decision makers without overloading them. In his examples, 4D-histograms[3] are deployed in the securities industry because this area is supplied with *huge amounts of data, the value of which declines rapidly with time*

[3] 3D space with time as a fourth dimension

(Wright 1995). Additionally, critical decisions have to be made based on this data in very short periods of time. In the literature, the stock market has remained a willing subject for business visualizations.

Business Visualization, or BizViz, is also deployed to improve quality and accessibility of risk management data. In this case, price samples are taken and put in a large warehouse. Insight is increased by projecting them in (three-dimensional) histograms. The big plus of such 3D histograms is the enormous amount of data they are able to display. For example Figure 2.13 summarizes a 60 page report all in one comprehensive 3D visualization.

An application area of business visualization which will play an important role in this dissertation is **management decision support**. The case study of Chapter 3 done at Asz/Gak, for example, mainly concerns the visualization of management information and its usage to support decision making. With the application of an interactive system, managers can evaluate and balance multi-dimensional factors, including risks and revenues of particular businesses. Senior managers can immediately see the effect of questions like *What if a business is delayed or stopped?* or *What are the pros and cons of supporting, or not supporting, a particular business?* (Gershon et al. 1997).

2.6 Summary and Conclusions

The need to *view* and *understand* complex, highly technical data by means of a visualization arose in sciences such as physics and chemistry. Scientific visualization is characterized as the transformation of symbolic data into visual structures. Research in scientific visualization has amongst others resulted in a collection of techniques to present information visually.

Information visualization is a relatively new research topic which has its roots in multiple, diverse disciplines such as mathematics, statistics, engineering, computer science and graphic design. Although most of this dissertation will look at visualization from a software engineering perspective, other viewpoints should not be neglected. Therefore, this chapter started with a discussion of the characteristics of visual information and, which is even more important, the design of useful and effective information.

Most differences between scientific and information visualization come from the fact that more abstract data, not having a clear connection with real life, is used in information visualization. Additionally, the audience using scientific visualization is more specialized and technically skilled than the diverse audience doing information visualizations.

Although (information) visualization is a relatively new discipline, a lot of lessons can be learned from visual information design. The purpose of visualization, just like the purpose of information design, is to provide users with knowledge to achieve their goals. It is important that in visualization, we

choose the appropriate visual mapping that highlights specific relationships in the data. As a consequence, it is impossible to create a 'best visualization.' Depending on the users and their tasks at hand, different visualizations are more expressive and effective to provide the user with the right knowledge.

CHAPTER 3

Management through Vision: a case study

Vision is the art of seeing things invisible.
Jonathan Swift

BizViz (Business Visualization) is a rapidly growing phenomenon in the realm of visualization. The main goal of BizViz is to support managers in their daily work of understanding forces within business processes and making decisions to control these forces. The case study presented in this chapter illustrates the added value of visualization at a social security institution (Gak) in the Netherlands. The prototype built, integrates both databases and simulation as information sources. This way managers can study the past and present as well as experiment with possible future situations.

The intended contribution of this chapter is twofold. First, it presents how visualization can (successfully) be applied in practice. It illustrates the added value of BizViz in the social security domain. As a second contribution, this chapter derives and discusses requirements that play an important role in visualization application development intended for decision-making support.

Structure This chapter starts with a general description of management information systems and what role visualization can play in order to increase the usability of such systems. After that, we focus on the current situation

at Gak Netherlands concerning the processing of benefit applications, which is the application domain of this case study. In the next sections, the visualization prototype created to support managing that process is discussed.

After the prototype was finished, we evaluated the concepts and actual implementation by visiting two Gak offices in the Netherlands. The results of this evaluation and a discussion of its outcome are given at the end of the chapter.

3.1 Business Visualization

Business Visualization is information visualization applied to the business domain to support decision making. By using business visualization effectively we can *accelerate perception, provide insight and control, and harness this flood of valuable data to gain a competitive advantage in making business decisions* (Gershon et al. 1997).

3.1.1 From data gathering to high-quality decisions

A couple of decades ago data was scarce. A lot of effort was needed to gather enough data about for example a market place or a production process. However, back then just like nowadays, decision-makers and managers had to be informed to decide which products to market or how to manage the business processes. Time has not stood still, especially with respect to data gathering. Due to new information technologies, data gathering has become very cheap.

Nowadays, we can get all the data we need. However, a different problem has entered the scene: *data overload*. There is so much data that the problem has shifted from data *gathering* to *separating* significant parts from the rest.

Why is good information in business so important? One explanation is that it is important because managers and decision-makers need the right information to make high quality decisions. In (Platinum 1998) a decision process is described, which is generally accepted as good business practice. It consists of the following steps:

1. Acquire (quality) raw data.
2. Combine and integrate the data to make useful information.
3. Analyze the information and make high quality decisions.

Thus, to make good decisions we first need good data. The difficult step, however, is the combination and integration of the raw data into useful information. The raw data must be processed in order to retrieve the right information for making decisions.

In the case of controlling a business process, managers monitor the business process and intervene whenever necessary. A recurring process cycle in the controlling process can be described as follows:

1. Do I have a problem?
2. Where does the problem manifest itself, what is its cause?
3. Think of a solution.
4. Evaluate the solution.
5. Implement the solution.
6. Check the effects of the solution.

Management processes are not linear in form. For example, when a solution does not have the foreseen effects, managers think of a different solution which then again will be evaluated, implemented and checked.

To support managers in selecting the right information and making good decisions, management information systems are deployed. For example in steps 1 and 2 of the process described above information from the business process under control is essential to localize problems and their causes.

Another part of the management process where information technology could be deployed is step 4. In this step, managers are trying to predict the effect of the new solution to the problem. By combining available data, a business prediction model and a simulation tool, it is possible to evaluate a future situation where a possible solution is implemented.

Basically two types of management information systems exist. The first type generates overview reports on a regular basis (e.g. weekly). Based on these reports, managers can see whether everything still conforms to the planning or whether they should intervene. A problem with this approach is that usually a large number of pages is generated while only a small part is useful in a specific situation. And finding the significant parts in the pile of paper is a time-consuming effort.

The second type of management information systems takes a different approach. Here, users can browse through the information in an interactive manner. The information can be filtered and sorted according to the manager's information requirements. However, the question remains how to identify the significant parts in the information space. Localizing known problems is easy, but discovering new problem areas remains cumbersome. The interactive nature of this approach only helps a little here. To further improve on the process of searching essential information, a different approach is necessary.

3.1.2 Visualizing management information

Problems with current management information systems are twofold. First, when producing reports it appears that we generate far too much information

while we only need a small part. A second problem of management information systems concerns the presentation of the information, which does not provide the users with enough insight to discover significant areas in the data.

To tackle the problems sketched above, we propose to use a visualization approach to access business data. And not as the final step, such as creating a pie chart from the numbers in a table, but right from the beginning. We want to start with a top-level visualization of the data which can be used to drill down to other visualizations or even to the bare numbers. We claim that by first providing a graphical overview and then zooming in on the details, users will be able to use the data more effectively and more efficiently. Shneiderman has called this approach the *Visual-information-seeking mantra:*

> Overview first, zoom and filter, then details on demand (Shneiderman 1998).

Walker (1995) has defined a framework for the conversion process from data to timely, informed decisions and increased, shared knowledge. Figure 3.1 (Walker 1995, Figure 1), which represents that framework, relates challenges within the conversion process with visualization.

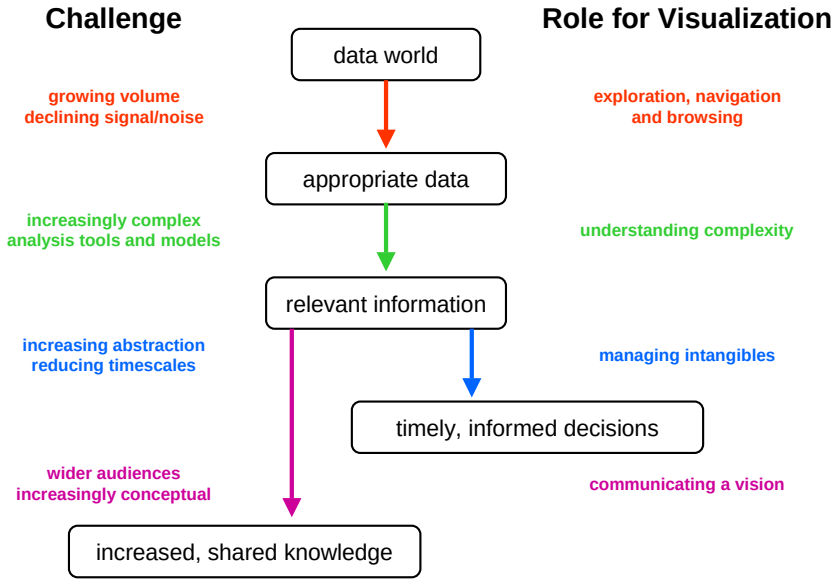


FIGURE 3.1: Conversion of data to informed decisions and shared knowledge. Courtesy of Walker

Walker describes the challenges in information visualization as follows:

> *The first challenge is locating and retrieving the relevant data, from a data world which is growing in size, but is also declining in average information content, as the provision of data becomes ever easier and cheaper.*

> *Analysing and interpreting data is the second challenge, and although sophisticated computer-based tools and techniques are available, they can often appear as inaccessible "black boxes" to the uninitiated. Even when the appropriate information has been extracted, two further challenges remain; the final conversion through to real benefits, expressed in our framework as better informed decisions and the sharing of increased knowledge. Difficulties in these areas center around the intangible nature of many of the issues and information under consideration, and hence the problem of establishing a common and comprehensive perspective. The four challenges are closely inter-related, and in any specific scenario the boundaries will be blurred if not invisible. However, the framework is nevertheless useful in clarifying some important issues and potential solutions. (Walker 1995)*

In terms of Walker's (1995) challenges for information visualization this case study focuses on three out of four challenges. First, it supports managers in exploration, navigation and browsing to find the appropriate data. Second, support to compare information helps to increase the understanding of complexity in order to find relevant information. Third, the goal of the case study is to improve the control and understanding of business processes in order to come to timely, informed decisions. Walker's last challenge, communicating a vision, is not so much covered by this study. It does become important, however, when we discuss collaborative visualization in Chapter 6.

3.2 Managing Business Processes at Gak NL

In The Netherlands, if people are unable to work, there are various welfare benefits they can apply for. The government does not handle the applications for benefits themselves, but gave this job to the so called Social Security Institutions. Gak Netherlands[1]is the largest social security institution in the Netherlands that processes applications for these welfare benefits.

3.2.1 The problem

The subject of this case study concerns the deployment of simulation and visualization to support business managers in planning and controlling the processing of benefit applications. To strengthen their position on the market, the Gak company wants to improve the efficiency of their production process without losing quality. The delivered products (in this case products are benefit

[1]The situation around social security is changing rapidly and significantly at the moment. At the time this case study was done, Gak Netherlands was still an independent institution that belonged to the *Gak groep*. Another member of the Gak group was ASZ, the ICT company that is responsible for the vast majority of information systems at Gak Netherlands. The case study that is described in this section is a collaboration between Gak Netherlands, ASZ and the Vrije Universiteit.

applications) are qualitatively good but the production process from the client arriving at the registration desk to deliverance (approval or disapproval of the application) takes too long. The norm production time for an application is in most of the cases 13 weeks, but due to unknown problems this norm is often exceeded. Obviously, the Gak company wants to find the production bottlenecks and their causes.

3.2.2 Current information systems

The source data coming from a number of production systems is collected in a single multi-dimensional data source as is often done in management information and decision support systems (Turban & Aronson 1998). Currently the Gak company uses Cognos' PowerPlay as its primary access to the collected information. PowerPlay is a program that can be used to filter out certain regions in a multi-dimensional dataset. Selection is achieved by drilling down on the various predefined dimensions. A dimension can, for example, be time-based or location-based. You can think of filtering on years, months or days or filtering on all offices, offices in a region or a specific office.

The dimensions available for drilling down are predefined, which implies that the possibilities are restricted to the dimensions chosen by the builders of the so-called PowerPlay cube. For example, the time that applications are already waiting at a particular phase in the process is not part of the PowerPlay cube. Although the information can be derived from the existing source data, managers are not able to view that information and, consequently, cannot use it to base decisions on.

The availability of time-based information, added to the visual exploration techniques enabled by visualizing data, inspired the idea of creating an information system that uses visual representations of the data to give a quick insight into problem areas.

3.2.3 Goals

The new visualization-based information system must support managers in making decisions. Typical questions that need to be answered by the system are:

Where can I expect capacity problems for the next weeks?

What kind of employee currently has too much work, which one has hardly anything to do?

Is the number of applications increasing, decreasing or stable?

In what stage is the majority of products and for how long are those products already in that stage?

Does the production process contain bottlenecks where products are always behind schedule?

Before we started the project we created a list of goals we wanted to achieve. The success of this case study is mainly determined by whether we succeed in accomplishing these goals:

- Solve the current problems: find the bottlenecks in the production process.
- Create a more **effective** management information system: managers will be able to get more information that answers their questions from the data.
- Create a more **efficient** management information system: managers will find what they need faster.

Although gaining insight in the current status of business processes and discovering trends of the past is important, users also expressed the wish to experiment with possible measures to avoid the recently detected bottlenecks. To allow for experimentation with control measures in possible futures, simulation is necessary. Simulation is a means to evaluate so called *what-if* situations. To increase the effectiveness of the visualization system, we decided to present both information sources, i.e. database and simulation, using the same visualizations. This allows managers to deploy the same views on past, present and future.

3.3 Visualizing Past and Present

To illustrate and validate our ideas about visualizing business information, we have built a prototype visualization system. The system was built in two phases. First, we concentrated on the current problems of the managers to indicate the bottlenecks in the business process. In this phase, we iteratively designed the visualizations which are shown in this section. In the second phase of the project, which is described in the next section, we created a simulation of the business process and used the previously designed visualizations to display the results.

The visualizations can be divided into quantity visualizations and capacity visualizations. Quantity visualizations provide insight into the number and status of applications for welfare benefits (usually called products) in the process. Capacity visualizations, on the other hand, represent to what extent the capacity, i.e. people, is employed.

3.3.1 Quantity visualizations

In quantity visualizations, two features are used to characterize the visualizations. First, the data source can contain either historical data about the **output** or up-to-date information about the current **throughput**. Second, we distinguish between a **global** product overview, i.e. all types of applications, as opposed to a **single** product type. This leaves us with four visualizations:

- Global product type overview of historical output.
- Global product type overview of the current throughput.
- Specific product type overview of historical output.
- Specific product type overview of the current throughput.

The prototype we created in this case study supports all four perspectives on the data. As an example of a specific, the visualization of the throughput of a specific product type (in this case the application for a disability benefit) is given in Figure 3.2. The visualization consists of two integrated parts, the business process structure and the throughput histograms. The connected circles represent the structure of the process, i.e. the phases an application has to pass through. A new application enters the process on the left and moves through stages from left to right. Each application is being processed during the depicted phases, such as intake and medical examination. When applications are dealt with, they leave the process structure again on the right-hand side. The percentages inside the circles reflect the amount of applications currently in that particular production phase that is ahead or on schedule. A higher percentage means a better timeliness of that stage in the process.

The histograms offer insight into the quantity and age of applications currently being processed. For example, the enlarged histogram (Figure 3.3) belonging to the middle phase (stage 158) in the process represents all products in stage 158 at that moment. The x-axis of the histogram is a timeline where each bar represents a single week in the production process. This means that the first bar of each histogram reflects applications in their first week of production, the second bar represents products in their second week, etcetera. The last bar summarizes all products that are more than 16 weeks in production and therefore seriously exceed the norm of 13 weeks.

In addition to the overall norm of 13 weeks to deal with an application, internal norms exist that indicate when an application should be in a particular stage of the total process. The histograms belonging to the stages reveal the number and status of applications currently in progress. To emphasize whether products are late or on schedule, we deploy colors for both the bars and the background of the histograms. Blue represents applications that are ahead of schedule. Green means on schedule, whereas purple is behind schedule. Red

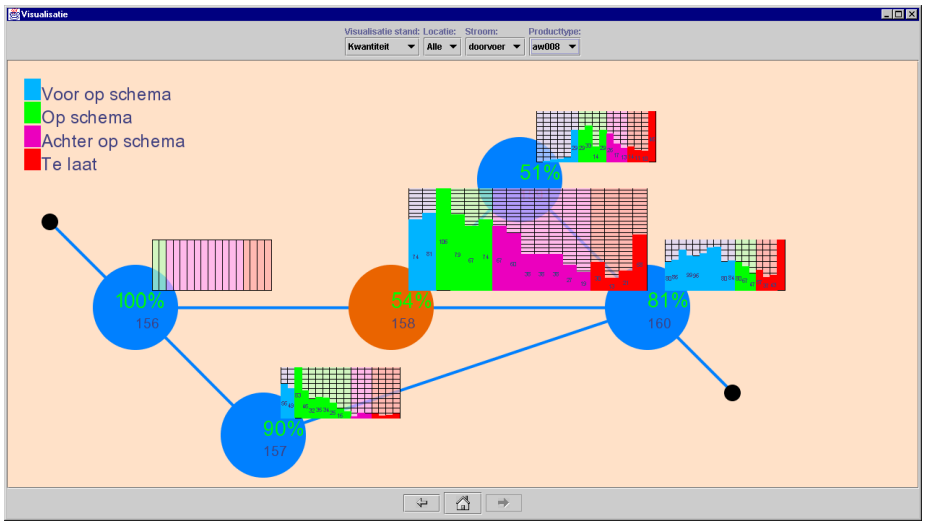


FIGURE 3.2: Throughput of a specific product type (AW008)

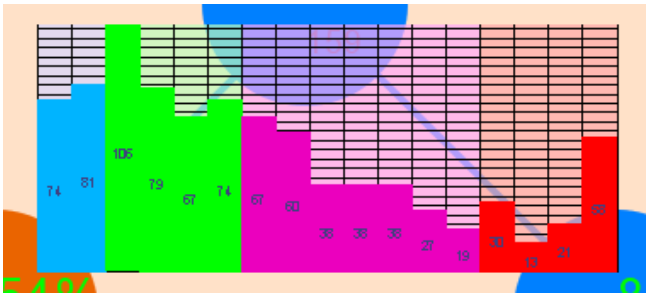


FIGURE 3.3: Histogram representing all products in a particular stage. Each bar represents a week in the production process. Color indicates whether applications are currently early (blue), on schedule (green), behind schedule (purple) or too late (red).

applications are even worse, because they are not only behind schedule but they are even late for meeting the total production norms of 13 weeks.

As an example, take a look at the enlarged histogram in Figure 3.3 again. There we see that, according to the norms, products should arrive at this stage in week 3 and leave again 4 weeks later (the green part). Additionally, we can determine that the norm time for the whole process is 13 weeks (the weeks before the background turns red).

It is interesting to notice what managers can derive from this visualization and how they might apply it to solve their problems. First of all, the percentages show that stages 158 and 159 are two possible bottlenecks with a timeliness of

respectively 54% and 51% while the other percentages exceed 80%. Other indications of possible problems are the red peaks at the end of the two rightmost histograms. These show that a relatively high percentage of applications is far too late at those stages of the process.

As a third observation, we can conclude that the pattern of the histograms appears regular —peaks in the green and decreasing on the right-hand side. This indicates a steady input of new applications without exceptional peaks as they may occur during particular periods of the year. However, the decreasing line to the right of the peak should be somewhat more steep to avoid timeliness problems; actually all applications should be processed before the background turns from green to purple.

The choice for traditional histograms and bar charts instead of more complex visualization forms was made because histograms were the best choice for our purposes. Ease of use and simplicity were important design requirements because the managers are unfamiliar with visualization and explicitly requested us to keep things simple. In related work, we see a similar quest for straightforwardness: *traditional bar charts work well for comparisons and are well understood, including by business people* (Zhang 1996).

3.3.2 Capacity visualizations

Whereas quantity visualizations concern the production or processing of applications, capacity visualizations are intended to represent to what extent the current capacity is used (or has been used). The visualizations do not contain quantitative information but percentages. Capacity visualizations are about people, the resources of this business process.

In line with the quantity visualization, we also distinguish between historical output and current throughput in capacity visualizations. The other dimension concerns whether we look at a single employee type or consider all types of employees. The two orthogonal dimensions determine the following four visualizations:

- Historical output of all employee types.
- Historical output of a specific employee type.
- Current throughput of all employee types.
- Current throughput of a specific employee type.

An example of a capacity visualization is shown in Figure 3.4. This figure shows a capacity usage visualization of past weeks (output of all worker types). The visualization is used to find out how employees have spent their time, in other words how the capacity (i.e. people) has been deployed during the last weeks.

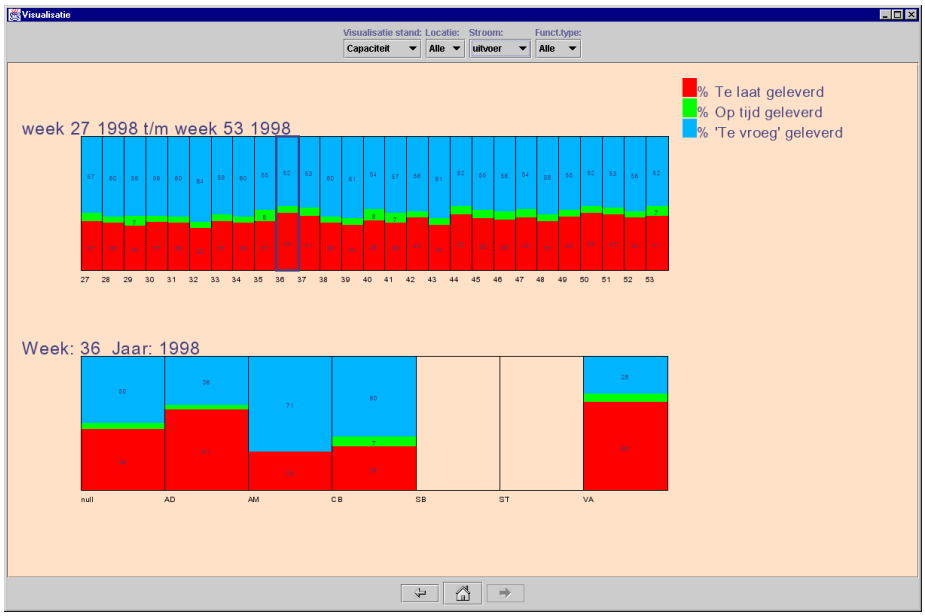


FIGURE 3.4: Capacity visualization of all worker types through time

Every bar in the upper histogram represents one production week, ascending from left to right. Every bar is divided into three parts, totaling 100 percent. The three colors (red, green and blue) in the bars represent percentages of products that are, respectively, behind schedule, on schedule and before schedule. If the red part of a bar represents 30 percent it means the employees worked 30 percent of their production time (for the week represented by the bar) on products which were behind schedule.

Selecting one of the bars in the upper histogram results in a second histogram at the bottom. This histogram is a decomposition of the selected bar in the upper histogram. It shows the usage of the various employee types for the selected week (*null* indicates that no employee type is known). Again the same colors are used to indicate how much time the specific employee types spent on processing products that are behind, on or before schedule.

In the example of Figure 3.4, the histogram suggests that the bad results are to a large extent due to the employees represented by the second bar from the left (AD) and the rightmost bar (VA) since their capacity is exceeded the most.

A second capacity example visualizes the load on the employees in the upcoming week (Figure 3.5). Each bar of the histogram represents a different type of employee. The red line represents the 100% capacity that is available for next week. If bars are higher than this 100%-line, the pile of work exceeds the available capacity. In Figure 3.5, the AD and CB (and to a lesser extend the VA) employee types are understaffed for the next week.

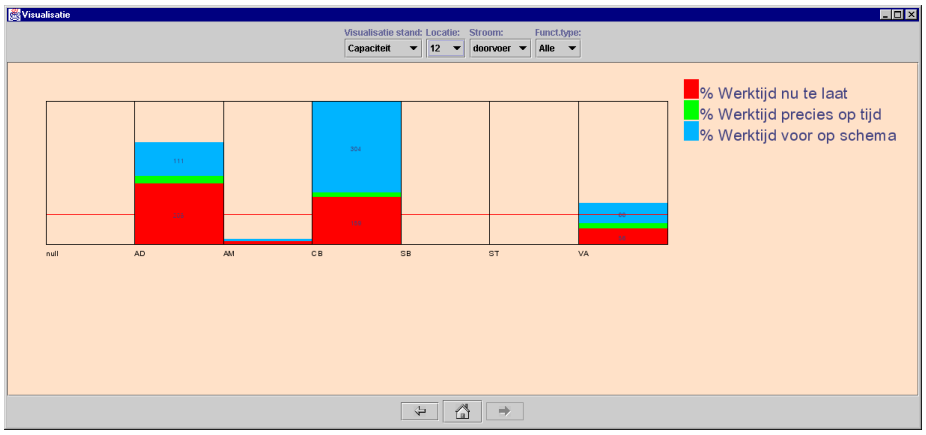


FIGURE 3.5: Capacity throughput visualizations show the AD, CB and VA bottlenecks

The colors of the bar represent the status of the work that is waiting to be processed. Red (bottom of each bar) represents applications that are already late, green (middle) is exactly on time whereas blue (top of bar) represents products that are ahead of schedule. In an ideal situation the 100%-line runs through the blue area, just above the green part, implying that all late and on schedule applications are processed during next week. In the case of Figure 3.5 however, the labor experts, represented by the leftmost column (AD), can only get rid of a small fraction of the late applications. The same problem occurs at the third (CB) and rightmost (VA) column. Summarizing, the visualization clearly demonstrates bottlenecks at the AD, CB and VA employee types, whereas the other types have a large overcapacity.

3.4 Visualizing the Future

Now that we can identify bottlenecks, a means of evaluating the effects of interventions is needed. To achieve this, simulation is needed.

3.4.1 Simulation

Simulation imitates the behavior of a system or process by applying probability calculus to a model of the system. The goal of simulation is to gain insight in the behavior of the system. Users can experiment by adapting simulation parameters and comparing the results of simulation runs.

In our approach, we tried to shorten the loop of making a model, run the simulation and analyze the results. We achieved this by integrating simulation,

interaction and visualization. While the simulation is running, we already visualize the results until that point. Additionally, users can interactively manipulate the settings and parameters of the simulation which are immediately reflected in the results of the simulation and, consequently, in the presented visualizations.

In the current software architecture, the simulation is a software component based on the JSim (Miller, Ge & Tao 1998) visualization library. The simulation produces information that can be visualized using the same visualizations as the ones used for the data coming from the databases. Interaction is achieved by a separate GUI that allows users to modify important parameters of the running simulation, such as the number of employees per type.

3.4.2 Trend visualization and interaction

Although the visualizations inherited from the representations of the databases are valuable to visualize the simulation, a new visualization has been added to better display the effect of interventions. Trend visualizations give a whole year overview of an application type or employee type.

For example, Figure 3.6 contains a capacity visualization of a specific employee type over the past simulated year. Each bar represents the work of a particular week compared to the available capacity (again the line indicates a 100% capacity). The colors of the different parts in the bars represent the capacity that is used to work on late (red) products or on applications that are behind schedule (orange), on schedule (green) and ahead of schedule (blue).

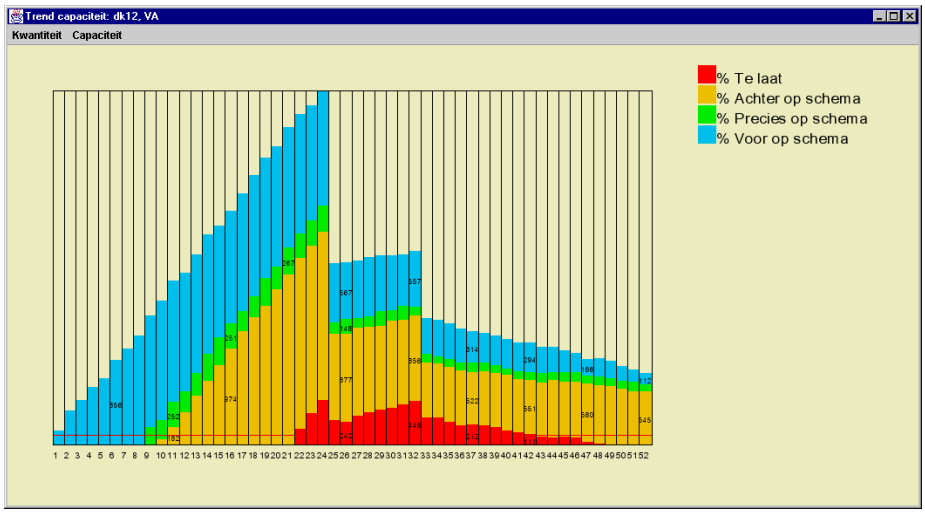


FIGURE 3.6: Trend visualization illustrates the effect of an intervention

The simulation started with an empty process in week 0. While the weeks advance, we can see that the number of applications pile up because the capacity of the employees is far too small. In week 25, the controller of the simulation increases the number of employees working on this stage in the process. Immediately, the height of the bar has decreased, because the amount of work is not so large any more compared to the available capacity represented by the 100%-line. However, the new number of employees is still not large enough to handle all the work, the height of the bars still increases over time. Only at week 33 when even more employees are added, we can notice a descending trend in the next weeks, indicating that the capacity is large enough to work away the backlog.

The integration of simulation into the visual information system clearly illustrates the power of combining simulation and visualization with user interaction. Based on experience gained by visualizing the static data in the database and knowledge of the business process, managers can experiment with all kinds of interventions. In addition to a single action, the effects of combined interventions can be evaluated and compared with complementary approaches. The set of available visualizations allows the experimenting manager to view the effects of interventions from different perspectives.

3.5 Evaluation of Concepts and Prototype

To evaluate the maturity of our visualizations and the usability of the developed prototype, we arranged two evaluation sessions at Gak offices in Rotterdam and Nijmegen. People who participated in the sessions are domain experts working at Gak Netherlands in the area of welfare benefits. The sessions started with a presentation about business visualization and an explanation of the prototype. After this, the participants had about 2 hours to work with the system. During this time, they were given some exercises to search for bottlenecks, compare the results of different products and investigate the capacity usage of the employees. Additionally, they were asked to fill in a questionnaire to provide us with feedback about the effectiveness of the visualizations and the usefulness and usability of the tool. At the end of the session, we presented the simulation extension.

Since the two evaluation sessions were done in small groups, with an equal amount of domain experts and developers, we had interesting plenary discussions which led to ample feedback. The combined results of the questionnaire and these discussions will be described now.

3.5.1 Benefits

The lack of information about the time that applications are already in a certain phase of the process appears to be an important omission in the current

information supply. However, the missing information can be derived from the available data sources and is available in the visualizations of the current throughput of specific product types. This extra piece of information was considered to be very useful in understanding delays in the process.

The participants of the evaluation were very positive about the usage of visualization to present the information. Without exception, they considered themselves visually oriented and had no problems with understanding the offered visualizations. Those present at the evaluation sessions agreed that the visualizations offered a better insight in current and past data. Additionally, they thought that it is easier to draw conclusions based on the visualizations than based on the currently used information system.

Another benefit concerns the availability of the easy access to up-to-date information. Our approach to disseminating the data and visualizations to the managers allows them to always inspect the latest information. This is a significant improvement over the weekly reports they receive now.

3.5.2 Shortcomings

During the evaluations at Gak Rotterdam and Gak Nijmegen, not only benefits were found. The shortcomings can be split in two categories: shortcomings of the concepts and shortcomings of the current prototype.

A problem with the concept of visualizing process data is the fact that the quality of the visualizations is highly dependent on the quality of the underlying data source. Visualization adheres to the slogan: *garbage-in, garbage-out*. The visualization nicely hides the fact that it is based on wrong or incomplete data and, subsequently, justifies wrong decisions.

Most of the shortcomings the managers came up with during the evaluation are only small problems, such as the absence of numbers on which the percentages are based. These shortcomings can easily be fixed in a next release of the system. A shortcoming with a larger impact is the request for an annotation possibility. This would be useful to evaluate the effects of taken measures at a later time. Additionally, it would help when somebody could read other people's opinions about certain phenomena in the data.

Another often requested feature was the ability to drill-down further on the data. Especially operational managers appeared to be interested in the functioning (in both positive and negative ways) of specific employees as compared to the rest of the group. The current implementation stops at the functionary-type level, but could be extended to represent data at the individual level since the necessary data is already available in the data sources. Whether this is wise thing to do, however, remains to be seen.

3.5.3 The simulation

In general, the participants of the evaluation reacted very positively towards the possibility to simulate decisions in the business process. However, the current intervention option (changing the number of available employees of a certain type) seems to be too limited. Other options such as setting priorities or skipping particular applications must be added to make the simulation extension useful in practice.

3.5.4 Discussion

In summary, everybody involved in the evaluation of the visualization application was pleased with the currently available visualizations. Especially, the current throughput of specific products and the capacity output visualizations appeared to be very effective. The idea of visualizing past, present and future is received with open arms. The current implementation of simulating business interventions, however, is not yet mature enough.

At the end of Section 3.2.3, we identified three goals that we wanted to achieve to make this case study successful. The first goal is to identify bottlenecks in the process to help solving the actual problems. The second and third goals are, respectively, to create a more effective (more useful information from the data) and more efficient (find what you need faster) management information system as compared to what is available now. Based on the evaluation, we can conclude that the bottleneck and efficiency goals are met: managers were able to identify bottlenecks in specific parts of the process and the visualizations helped them to understand what is going on more quickly.

The goal of creating a more effective management information system in which managers are able to get more information that answers their questions is only partly achieved. On the negative side, the managers are not able to retrieve all the information that they could get from the current information system. Especially, detailed numeric data is unavailable in the current system. On the positive side, however, participants indicate that new and very useful information is made available that was not accessible before. For example, insight in the age of applications at particular stages in the process is a new and important piece of information that is useful to control the allocation of the available capacity of employees.

3.6 Discussion and Issues Raised

The motivation to perform a series of case studies at Gak Netherlands was to illustrate and prove the added value of business visualization to support decision making. In addition, performing a BizViz case-study in a real-world

situation will lead to a better understanding of requirements and features that play a role in visualization application development.

Below the most relevant issues raised during the case studies are given.

- **Multiple visualizations**
  The case study described here clearly demonstrates that a single visualization is not sufficient to meet all information needs required by its users. For a visualization to be useful, it has to be related to the tasks and goals of the people using it. In other words, visualizations only help when they provide useful information to support people in their tasks such as decision making.

- **Derivation of new information**
  Information that appears to be crucial for good decision making is not always directly available from the data sources. Relations within the raw data are often more valuable than the data itself. For example, information about how long applications are already stuck at a particular stage is not directly saved in the source databases. The information can, however, easily be derived and presented to the user.

- **Iterative approach to create effective visualization**
  Although not covered in the discussion of the case study given here, we experienced that creating straight-forward, effective visualizations cannot be done out of nothing. We discovered a close resemblance to engineering software for a client who does not really know what he or she wants. An approach taken to tackle that problem in software engineering is an iterative approach with a lot of end user interaction, e.g. in *Rapid Application Development* (Martin 1991). In the case studies performed at Gak Netherlands this turned out to be a good choice for visualization development too. We collaborated closely with the people actually using the information and iteratively created the questions we wanted to solve as well as the visualizations that are intended to answer to those questions.

3.6.1 Organizational forces

During the project we encountered some unexpected delay due to problems with the available data and the organizational process of gathering information. It appeared that each manager is allowed to adapt the process according to his or her wishes. Additionally, employees do not like to write down how long they have been working on a specific application. These and other problems are the main cause of the high noise ratio in the data which, consequently, influences the reliability of visualization and simulation.

We reported these problems back to the management of the company and to the project team responsible for re-engineering business processes and the IT

support thereof. In the near future, they will decrease the freedom of each manager by making procedures more rigid. Additionally, a workflow system will keep track of the status and phase of each application, alleviating the employees of these boring tasks. We expect that the proposed measures will improve the quality of visualization and simulation. At the same time the practicability of our prototype as a means to control processes increases.

3.7 Summary and Conclusions

This chapter presented a case study done at Gak Netherlands to illustrate the usage of simulation and visualization to control business processes in the domain of social security. The project was done in two phases. First, we iteratively designed visualizations to understand recorded historic data about the past and present. After that, we created a simulation component to experiment with possible futures to solve problems in the current production process.

Evaluation showed that visualization could indeed aid in better understanding of the available information. Consequently, visualization leads to more informed decisions. The usefulness of the current prototype could be increased further by allowing quick access to up-to-date data, extending the intervention evaluation possibilities and by adding support for collaboration. Issues raised during the case studies are used as the basis for requirements of a software architecture for information visualization.

As a conclusion, we state that interactive simulations and visualizations are a powerful means to control business processes. Especially, an integrated solution combining the advantages of retrospection and experimentation, allows decision makers to discover trends in the past, monitor the current situation and predict possible futures.

CHAPTER 4

Visualization Models: theory and practice

Form is the external expression of internal content
Wassily Kandinsky.

Progress in science is often based on knowledge transfer from related work. This way, one can build on previous experiences and make progress. In the case of information visualization two important sources of knowledge exist: theoretical models on the one hand and practical projects on the other hand.

The goal of this chapter is to provide an overview of the most influential visualization models, both in theory and in practice. Theoretical models describe the process of visualization as the transition from raw data to visual representations. These models are necessary to better understand the visualization process. Additionally, they provide a framework for developing software support for visualization.

The second part of the chapter discusses visualization tools, ranging from built-in visualization functions, via visualization component libraries to research projects. There, we will provide a broad overview of the spectrum of computer-based visualization support and discuss the strong and weak points of state-of-the-art visualization tools.

4.1 Visualization Models

This section discusses four theoretical models designed to describe the process of visualization. The *visual taxonomy* and *visualization pipeline* come from the realm of scientific visualization whereas the *visualization reference model* originates in the study of information visualization. The last approach given in this section is a formal approach describing the mappings from abstract data to visual information structures.

4.1.1 Visual taxonomy

The first and probably most general model discussed here is based on the process of scientific visualization as described by McCormick et al. (1987). The rationale of the *visual taxonomy* is the distinction between **symbolic structures** or structural data on one side and **visual structures** on the other side. Visualization itself is then considered as the **mapping** from symbolic to visual structures.

In scientific visualization structural data is the result of computational methods, such as finite element analysis or numerical analysis. Another possible data source is measured data, for example coming from CT and MRI scans, or from the financial system at a stock market. Before data is visualized, it can be transformed to other non-visual data. This is useful when computed data is not in the appropriate format or whenever statistically derived information is necessary.

The process of visualization, called **mapping** in this model, transforms the symbolic structures to visual structures which are, accordingly, presented to the user. When the visualization is effective, a user might gain enough insight to influence the process of data computation or measurement. This is often referred to as **analysis** or **computational steering**.

Figure 4.1 visualizes the two types of structures and the transitions in the visual taxonomy. The figure shows that, while mapping and analysis go from symbolic to visual structures and vice versa, transformations manipulate structures in their own domain. The model discussed next, the visualization pipeline, approaches visualization by taking a closer look at the mapping transition represented by the rightmost arrow in Figure 4.1.

4.1.2 The visualization pipeline

The model underlying the visualization pipeline (Schroeder et al. 1996) focuses on the (one-way) **transformation** from source data to a visual representation. The visualization pipeline model is inspired by the UNIX pipe and filter architecture used by shell and command line tools. The processing is accomplished by **filters** which perform transformations on the data. Filters communicate by

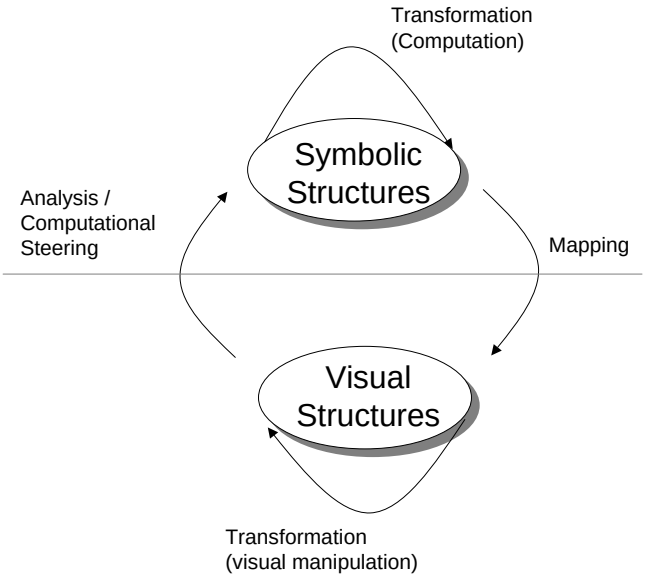


FIGURE 4.1: Visual Taxonomy

means of **pipes** which store the output coming from filters until it is processed by the next filter in the pipeline.

Filters with no input are called **sources** and filters with no output **sinks**. Figure 4.2 contains a schematic overview of the visualization pipeline consisting of a data source, two filters to transform the output to visual data capable of being displayed by the display sink.

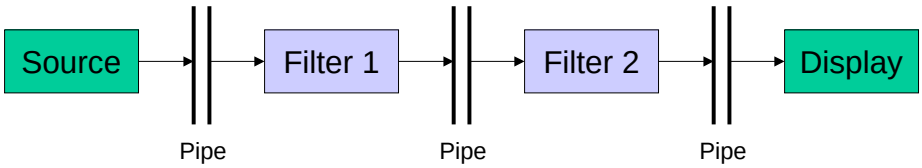


FIGURE 4.2: Visualization pipeline

To let filters communicate with each other, the input and output of two linked filters have to be type compatible. Usually, visualization systems define a limited number of data types to cover most forms of data but, at the same time, make filters as inter-operable as possible.

A nice example of a pipeline-based visualization application is AVS Express. The product consists of a visual programming environment and a large library of processing components (sources, filters and sinks). Users can create visualizations by linking components as illustrated in Figure 4.3. The pipes are

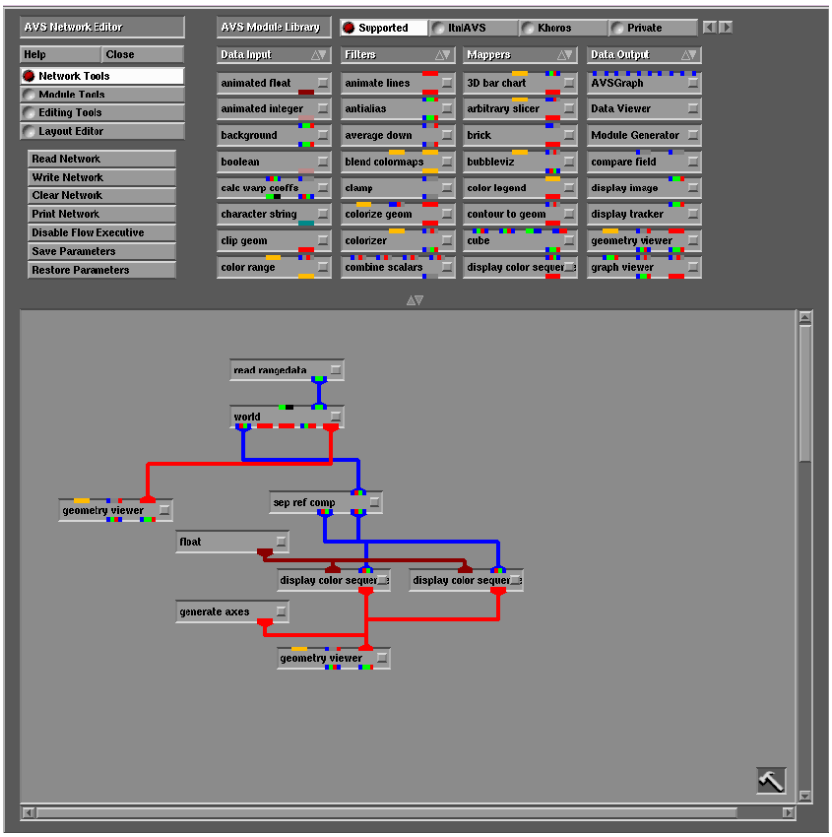


FIGURE 4.3: AVS Express embodies the visualization pipeline in a visual programming environment. Courtesy of Advanced Visual Systems Inc.

represented by fat lines, where the color of the line represents the type of data flowing over the connection. In line with the pipeline approach only components which are type compatible (recognizable by the color of the input and output ports) can be connected.

4.1.3 The visualization reference model

Both previous models, the visual taxonomy and the visualization pipeline, describe visualization as mappings from symbolic data to a visual form. Card et al.'s (1999) visualization reference model combines these transformations with the notion of human interaction, as shown in Figure 4.4 (Card et al. 1999, Figure 1.23).

Just like the other models, the reference model starts with raw data that is,

subsequently, transformed into a graphical presentation. First, using the terminolgy from (Card et al. 1999, p.17) **data transformations** map **raw data** in some idiosyncratic format into data tables. **Data tables** have a relational structure and include metadata describing the rows and columns. At the core of the reference model, **visual mappings** transform data tables into **visual structures** describing the graphical properties of the visualization. As a final step, Card et al. (1999) mention **view transformations** that generate views of the visual structures by specifying position, scaling, clipping, etcetera.

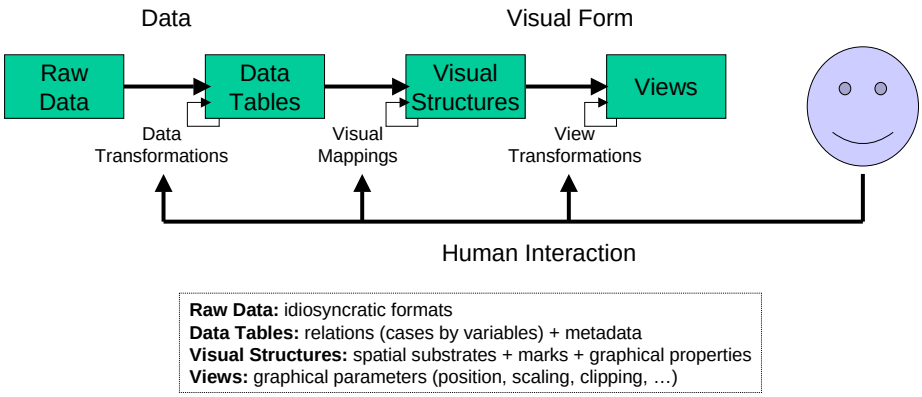


FIGURE 4.4: The visualization reference model adds human interaction as a means to manipulate transformations and mappings. Courtesy of Card et al.

In the reference model, human interaction takes place at three different stages in the transformation process. For example, by modifying the data transformations, data can be ordered or classified differently. By changing the visual mappings users select different visual forms to view the same data of the data tables. Finally, altering view transformations include simple interactions like scrolling or modifying the viewpoint in a 3D world. However, location probes, such as brushing and distortion control (e.g. a bifocal lens) also belong to the view transformation.

Example

Since the visualization reference model plays an important role in the remainder of this thesis, we now give a small example to illustrate how raw data transforms into tables, visual structures and, finally, into a representation. The example concerns the visualization of how students spend their time at the university. Without claiming that this is a useful or effective visualization, the example illustrates the transformation of data (both content and structure) from a raw data source into a visualization. The intended visualization presents what percentage of their total time students invest in studying particular topics.

TABLE 4.1: Data transformations

Student number	Topic	Time spent
1234	SE	16 hrs
1234	UID	12 hrs
1234	OOP	24 hrs
…	…	…
1444	AI	100 hrs
1444	SE	10 hrs
…	…	…

(a) All data about how students spend their time is gathered into a single table

Student number	Topic	Time spent
1234	Total	52 hrs
1444	Total	110 hrs
…	…	…

(b) New rows are added that sum up all information about a single student

Student number	SE	UID	OOP	AI	Total
1234	16 hrs	12 hrs	24 hrs	0 hrs	52 hrs
1444	10 hrs	0 hrs	0 hrs	100 hrs	110 hrs
…	…	…	…	…	…

(c) The transformed table structure contains a single row for each student

Student number	SE	UID	OOP	AI	Total
All	26 hrs	12 hrs	24 hrs	100 hrs	162 hrs

(d) A row containing the sum of all students' data is added

Student number	SE	UID	OOP	AI	Total
1234	31 %	23 %	46 %	0 %	52 hrs
1444	9 %	0 %	0 %	91 %	110 hrs
…	…	…	…	…	…
All	16 %	7 %	15 %	62 %	162 hrs

(e) The data is transformed from quantity to percentages

The first step in the process gathers all information into a single table of which the columns describe the student number, the topic and the time spent on that topic measured in hours. The rows in the table contain all the measured facts. Table 4.1(a) contains an example of the resulting table.

To calculate what percentage of their time students have spent on particular topics, we have to know their total time invested. Therefore, we have to add derived information about the total time spent by each student. Table 4.1(b) contains the rows that have to be added to our table.

The next step in the data transformation process, modifies the structure of the table. The topics are promoted to properties and, therefore, appear as columns in the table. The new table contains only a single row for each student. This row describes how much time has been spent on the different topics. Table 4.1(c) illustrates the transformed table of our example data.

After this, a single row containing the sum of all data is added as illustrated in Table 4.1(d). This row can later on be used to say something general about the average student.

The final data transformation step in this example converts the contents of the table. The unit of the rows describing particular topics, such as SE or UID, are converted into percentages of the total time spent. The results of this modification are given in Table 4.1(e)

Finally, the information available in Table 4.1(e) is applied to a visual mapping and view transformation resulting in the pie charts of Figure 4.5. The number of pie charts equals the number of students plus one for the average of all students. Using the pie charts we can quickly see how the average, or a specific, student splits up his or her time over the offered courses.

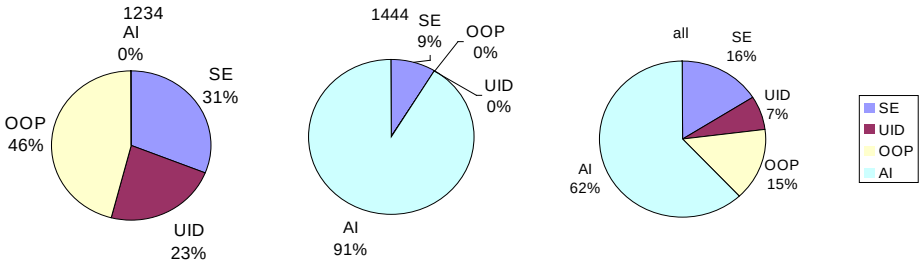


FIGURE 4.5: Pie charts visually represent the information of Table 4.1(e)

4.1.4 A formal framework for visualization

The fourth and last description of the visualization process given here is a formal approach. The discussed formal framework for data visualization is based on (Dastani 1998). It considers visualization as a (structure-preserving) mapping between two relational systems. The first relational system describes the

structure of the data and the second system describes the perceptual structure of visual patterns. Any mapping between those systems will provide a visualization. However, only a limited subset of all possible mappings represents effective visualizations.

The purpose of describing the source and destination relational systems and the mapping between them formally is to prove characteristics of visualizations with respect to effectiveness[1]. A visualization is considered to be effective if the *intended* data relations are visualized by structurally identical *perceivable* relations. In other words, all available and useful information (and nothing more) is visible in the graphical representation.

The formal framework comprises a data relational system, a visual relational system and the mapping between those systems. Both relational systems consist of a set of entries and a set of relations defined on that set of entries. One can think of a relational system as a table without order but with meta-information describing relationships between the rows of the table. More formally, Dastani (1998) defines the data and visual relational systems as follows:

> **Definition 4.1.1** *Let DE be a set of data entries (n-tuples of data-attribute values), and let $DR_1, \ldots, DR_m$ be partial relations that are defined on DE by means of the data attributes involved. Then, an annotated data table is defined as a relational system $< DE; DR_1, \ldots, DR_m >$. Such a relational system will be called data system.*
>
> *Similarly, let VE be a set of visual entries (n-tuples of visual attribute values), and let $VR_1, \ldots, VR_m$ be partial relations that are defined on VE by means of the visual attributes involved. Then, an annotated visual table is defined as a relational system $< VE; VR_1, \ldots, VR_m >$. Such a relational system will be called visual system. (Dastani 1998, p.166).*

Visualization itself can now be defined as the combination of a mapping between the domain elements of the relational systems and a one-to-one mapping between their relations. For a visualization to be effective, an **isomorphism** (structure-preserving mapping) is required between the relational systems.

> **Definition 4.1.2** *An isomorphism between two relational systems $< \Delta_1; R_1, \ldots, R_n >$ and $< \Delta_2; S_1, \ldots, S_n >$ is a one-to-one and onto mapping between Δ_1 and Δ_2 and a one-to-one mapping between relations R_i and S_i (for $i = 1, .., n$) which satisfies the following condition:*
>
> *If a relation R_i holds between two elements of Δ_1, the corresponding relation S_j holds between the corresponding elements of Δ_2 and if R_i does not hold between two elements of Δ_1, the corresponding relation S_j does not hold between the corresponding elements of Δ_2. (Dastani 1998, p.171).*

[1] What Dastani (1998) calls effectiveness is more or less what Mackinlay (1986) calls expressiveness (see also Section 2.3.2 on page 22). Despite the fact that this may cause confusion, I decided to keep the original terminology.

By requiring the visualization to be isomorphic, it is guaranteed that each data entry is represented by a visual entry. Additionally, whenever there exists a relation between two data entries, their corresponding visual entries are related to each other by a structurally identical relation.

Example

To clarify the usage of the formal framework, consider the following dataset describing the annual sales of a company. The set consists of *(year, value)* tuples. Since a set does not have an intrinsic order, we define the order of the elements after the natural ordering of the year attribute. Thus, the data system consists of the dataset D and the E_D relation as follows:

$$D = \{(1997, 2000), (1998, 4000), (1999, 8000), (2000, 16000)\}$$
$$E_D : D \times D \text{ is defined by}$$
$$\forall (a, b), (c, d) \in D : ((a, b), (c, d)) \in E_D \iff a < c$$

The dataset D describes 4 data tuples in the form of $(year, value)$. The earlier relation E_D defines that a tuple $(year_1, value_1)$ is earlier than $(year_2, value_2)$ whenever $year_1 < year_2$.

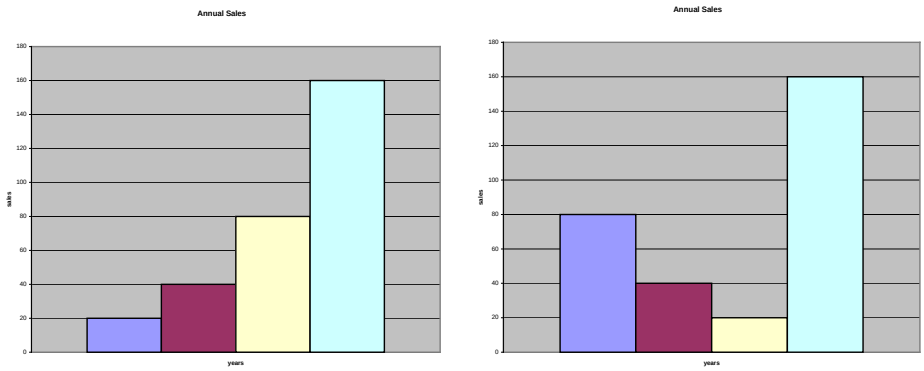


FIGURE 4.6: Left: Since the structure of the data source has been maintained, this bar graph is an effective visualization. Right: A non-isomorph mapping mangles important relations between the individual elements.

Two possible visualizations of dataset D are give in Figure 4.6. The left histogram is an effective visualization of the data system $< D; E_D >$ because the elements and ordering are preserved. The histogram on the right of the figure, however, is not isomorph with the original data system. The year-ordering relation E_D is not preserved and, therefore, the visualization is not effective. The definition of the visual system describing the left histogram is as follows.

$V = \{(1, 20), (2, 40), (3, 80), (4, 160)\}$
$L_V : V \times V$ is defined by
$\forall (a, b), (c, d) \in V : ((a, b), (c, d)) \in L_V \iff a < c$

The visual system consists of data tuples representing the bars in the histogram and a relation describing the left-to-right order of the histogram. The visual entries describe the position and height of the bar: $(x, height)$. The L_D relation defines that a bar is to the left of another bar whenever its x-position is smaller. Defining the isomorphism between the data system $< D; D_E >$ and the visual system $< V; V_E >$ is now trivial.

4.2 Visualization models in practice: tools and research

Visualization models do not only occur in descriptive theories of how the process of visualization takes places. On the contrary, visualization models are an important ingredient of tools and applications that perform visualizations. The choice for a particular visualization model even determines the richness of the visualization environment. For example, building a tool that supports interactive visualization around a visualization model that does not take interaction into account is a difficult job.

To structure the discussion of the practical side of visualization models, we have grouped the tools, environments, etcetera in four distinct sets. The first group, that we call embedded visualization, consists of products that are not specifically built for visualization but contain visualization functionality as a feature. The second group, comprises general visualization creation and presentation tools intended for end users. The third group consists of software libraries and component repositories which can be deployed to build new visualization applications. And finally, the fourth group comprises research projects that focus on a particular aspect of visualization systems.

Table 4.2 contains a list of all visualization environments that are discussed in the remainder of this section. We believe that this set of tools and research projects covers a broad area of practical visualization in practice. The choice for this particular set of environments instead of a different one is based upon the author's experience with most of these tools.

4.2.1 Embedded visualization

Visualization is increasingly found in existing data management or generation tools. For example, databases, spreadsheets, statistical packages, simulation

TABLE 4.2: Overview of the visualization environments

Visualization environment	Description	Application Domain	User group	URL
Microsoft Excel	Built-in visualization capabilities (charts) to present spreadsheet data.	All domains for which a spread sheet is applicable.	End users.	www.microsoft.com/office/excel/
Systems Modeling's Arena	Simulation tool with built-in visualization and animation capabilities.	Special editions available for specific domains, such as business and call-center edition.	Simulation developers and end users.	www.sm.com
AVS Express	General-purpose visualization product.	Scientific and technical visualizations.	Developers and end users including scientists and researchers.	www.avs.com
IBM Open Visualization Data Explorer (OpenDX)	Open source visualization framework (GUI tool and API).	Scientific and data visualization.	Visualization developers and end users.	www.research.ibm.com/dx/
Visualization Toolkit (VTK)	Open source C++ library with Tcl, Java and Python bindings. Optional commercial support.	3D graphics, image processing, scientific visualization.	Programmers and visualization developers.	www.kitware.com/vtk.html
AVS OpenViz	Commercial component suite.	Business Visualization.	Visualization developers.	www.avs.com
muSE (multi-dimensional user-oriented synthetic environment	High-end visualization environment that enables multi-user, multi-platform, and multi-sensory applications.	Multiple, for example aerospace, database, and oil & gas.	Visualization developers.	www.musetech.com
Computational Steering Environment (CSE)	Research project, software architecture and implementation.	Interactive data visualization of integrated simulations.	Software developers.	www.cwi.nl
CMU's Visage	Research project, prototype software environment.	Information visualization, information-centric paradigm.	End users.	www.cs.cmu.edu/~sage/visage.html www.maya.com/visage/
Nasa's Visual Analysis Graphical Environment (Visage 3.0)	Visualization development environment, framework of APIs (free binaries).	Information visualization.	Visualization developers.	tidalwave.gsfc.nasa.gov/avatar/projects/visage/

tools and so forth contain more and more visualization capabilities. To guide our discussion of embedded visualization, we will use two well known examples: Microsoft Excel and Systems Modeling's Arena.

Excel Excel (Microsoft 1999) is a spreadsheet program that has some basic visualization capabilities: data available in the rows and columns of the spreadsheet can be shown in a graph. To facilitate the visualization process, Excel contains a wizard (Figure 4.7) that helps the user to create the graph that represents the spreadsheet. The underlying visualization model in Excel is extremely simple. The source data is converted into a visualization representation in a one-way mapping. The model is therefore comparable with a simplified version of the visual taxonomy. The difference is that going back from the graph to the data is not supported. The visual mapping, however, can be adapted easily.

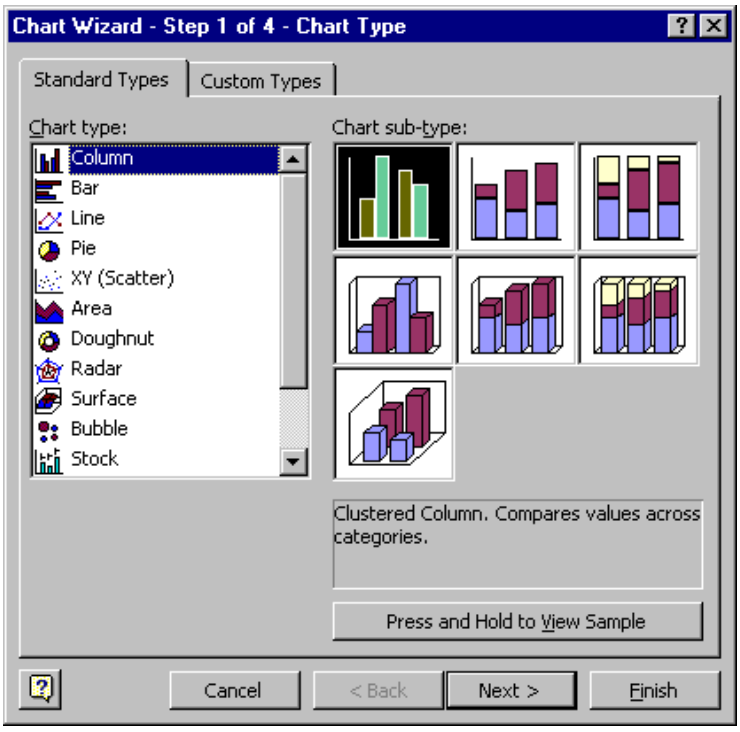


FIGURE 4.7: Microsoft Excel contains a wizard that helps the user to create a graph visualization of spreadsheet data. Courtesy of Microsoft Inc.

Arena With Arena (Bapat, Drake & Sadowski 1998), Systems Modeling Corporation provides an extensive suite of modeling, simulation and animation

tools. The collection consists of a core simulation engine and a number of extensions to cover a broad range of application domains. Figure 4.8 shows an example of the business edition of Arena. The figure shows how Arena integrates modeling, simulation, visualization and animation. To create such an animated visualization, the user first models the business process by specifying and connecting the steps within the process. The result of this modeling step is shown in the top of the figure. After that, visualization or animation elements can be added such as a bitmap representing the activity of the receptionist or a graph showing the current number of applications in progress. By pressing the *run* button, Arena starts the underlying simulation and animates the visual elements (including the business model) to provide feedback on the simulation.

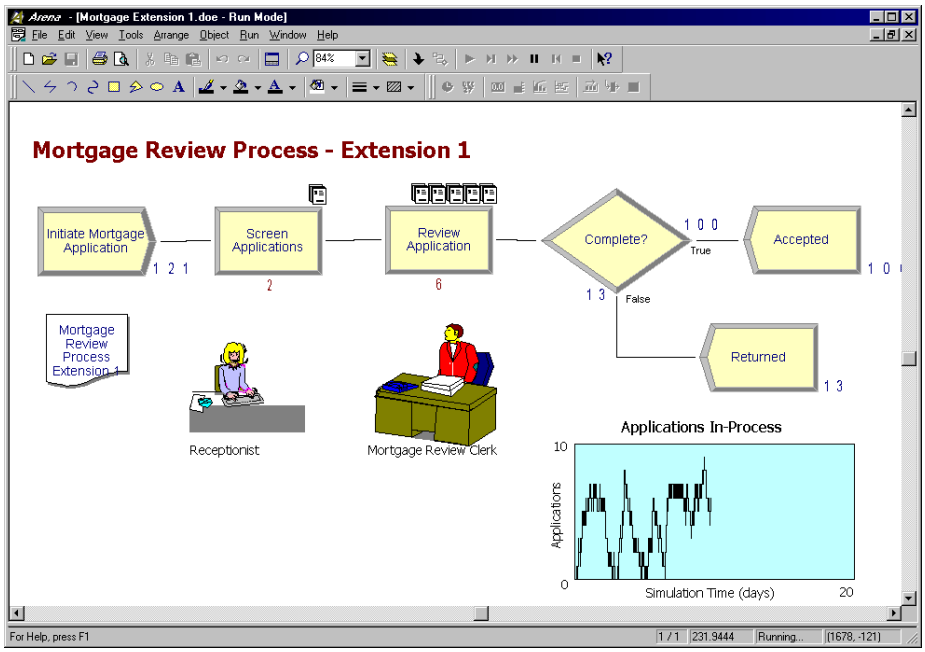


FIGURE 4.8: Arena combines modeling, simulation, visualization and animation. Courtesy of Systems Modeling Corp.

The visualization model deployed by Arena is hidden within the Arena package itself. Modeling, simulation and visualization are all tightly integrated into the application. The choice for a tight coupling has the advantage that creating a visualization is relatively easy. For example, when you use Arena to model your business process you get an animated version of your model for free. However, decoupling the simulation data from the built-in visualization is cumbersome. Using more sophisticated visualization tools on the basic simulation engine is not directly supported.

Discussion: tight versus loose coupling The issue of tight versus loose coupling is important in visualization. Moreover, we see tight coupling as one of the problems that we want to solve in this research project (see also Section 1.1 on page 2). Embedded visualizations are usually tightly coupled. This is fine for basic visualization tasks. However, for more elaborate and powerful visualizations a looser coupling allows for more flexibility regarding the deployed visualization mechanisms.

4.2.2 General-purpose visualization tools

The second group discussed in this chapter comprises visualization creation and presentation tools intended for end users. In this category, users, such as scientists and business managers, can visually examine data sets which are imported into the visualization tool. In addition to this, developers can often write their own components that can be plugged into the tool. Members of this group are AVS Express and IBM Open Visualization Data Explorer (OpenDX).

AVS Express AVS Express (Westmacott 1997) is a general-purpose visualization environment. The visualization model underlying AVS Express is the visualization pipeline. AVS Express contains a collection of filters (called modules) that can be deployed to manipulate, filter and present data. The modules are connected via communication links. As already shown in Figure 4.3, AVS Express has a visual programming environment to wire a visualization by connecting data sources, filters and views. Developers can extend the capabilities of AVS Express by creating their own modules and deploying them as filters in the visual programming environment.

OpenDX A similar approach is taken by the IBM Open Visualization Data Explorer (OpenDX). OpenDX provides the user with a range of analysis, manipulation, rendering and animation modules within a graphical development environment. Figure 4.9 contains a screenshot of OpenDX in action. The figure shows the deployed components and connections in a visualization of the ozone layer. Users can adapt the visualization process by modifying the visualization pipeline shown on the right-hand side of the figure. Settings of specific visualization components can be controlled through separate GUI controls. Modifications in the visualization pipeline or one of its components are immediately reflected in the image shown on the left side.

Discussion: extensibility of visualization environments Since the visualization process often requires specific analysis, manipulation, or visualization primitives, extensibility is an important requirement for visualization development environments. Therefore, most tools include the possibility to plug-in your own components. As an example of how powerful the addition of new

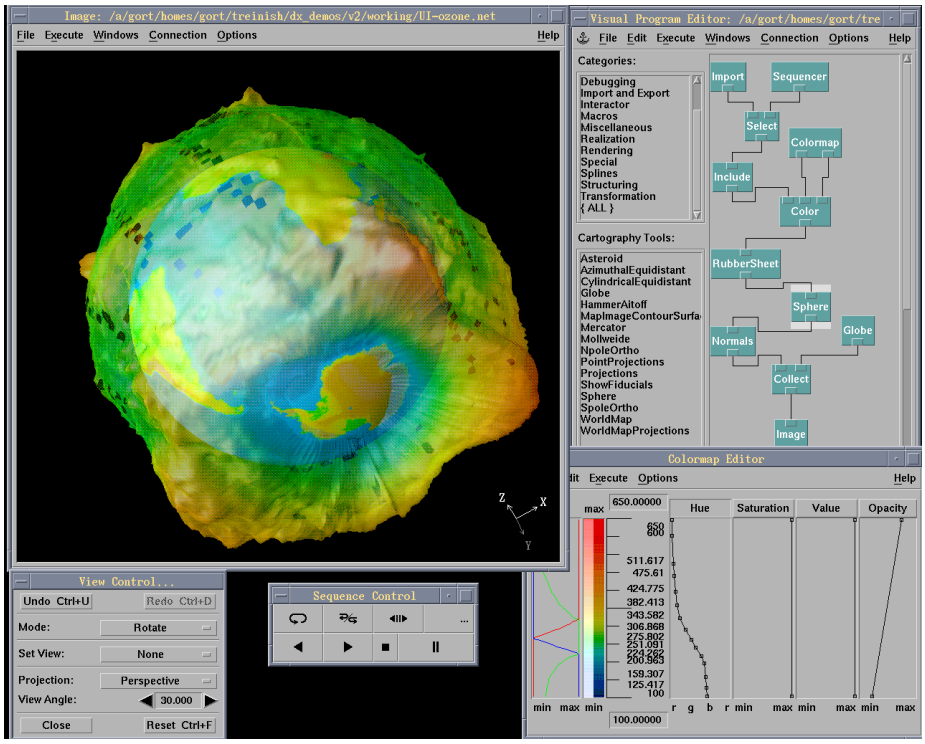


FIGURE 4.9: IBM Open Visualization Data Explorer contains a visual program editor to develop visualizations. Courtesy of IBM Research.

component can be, we will show how AVS Express can be extended to support collaborative visualization.

Although AVS Express itself does not include support for collaboration, the San Diego Supercomputer Center (SDSC) developed modules to support collaborative behavior (Elvins & Johnson 1998). By connecting the visualization components forming the visualization pipeline by means of a collaboration server, as illustrated in Figure 4.10 (Elvins & Johnson 1998, Figure 3), a fully shared control visualization can be achieved.

4.2.3 Visualization component libraries

The third group we discriminate comprises software libraries or component repositories which can be deployed to build visualization applications. Usually, a development tool or language is provided to simplify this job. The relation with the previous group is strong. However, in the visualization tools group, the visualization development tools are considered the most important element of the products, whereas in the component libraries group, the indi-

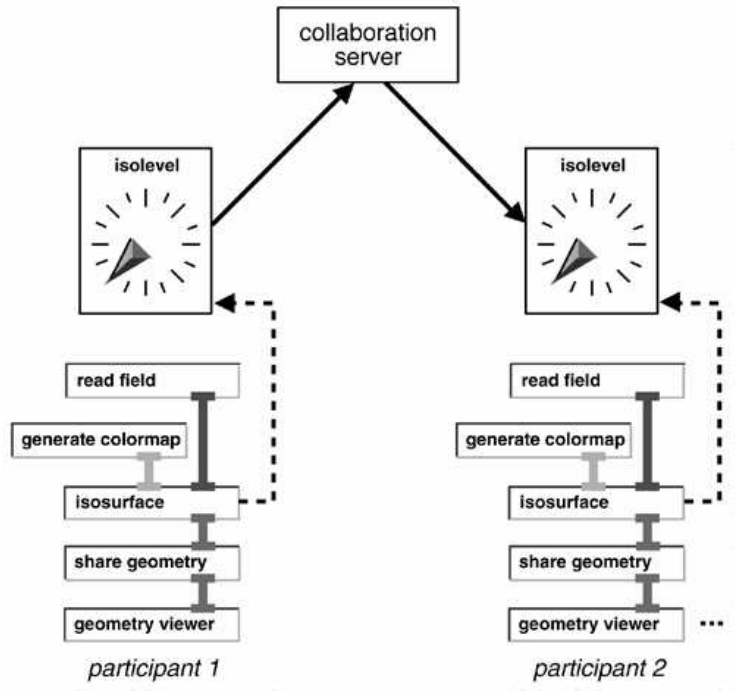


FIGURE 4.10: Collaborative AVS extends AVS Express with collaboration components. The collaboration server ensures that changes in a module at one machine are reflected in the corresponding modules on the other machines within the collaboration session. Courtesy of Elvins and Johnson.

vidual components are the main ingredient. Members of the software library group are the Visualization Toolkit (VTK), OpenViz and MUSE.

VTK The Visualization Toolkit (Schroeder et al. 1996, Schroeder, Avilla & Hoffman 2000) is a software system for 3D computer graphics, image processing, and visualization. VTK is based on a C++ class library, but has bindings to other languages including Tcl/Tk, Java, and Python. The underlying visualization model is based upon the visualization pipeline. Visualization developers build the filters and connect them using VTK's connection mechanisms.

OpenViz OpenViz (Advanced Visual Systems 1999) is a collection of Java and COM components especially aimed at business visualization. Developers can deploy the OpenViz components to build custom visualization applications. OpenViz components range from data manipulation to interactive viewer components.

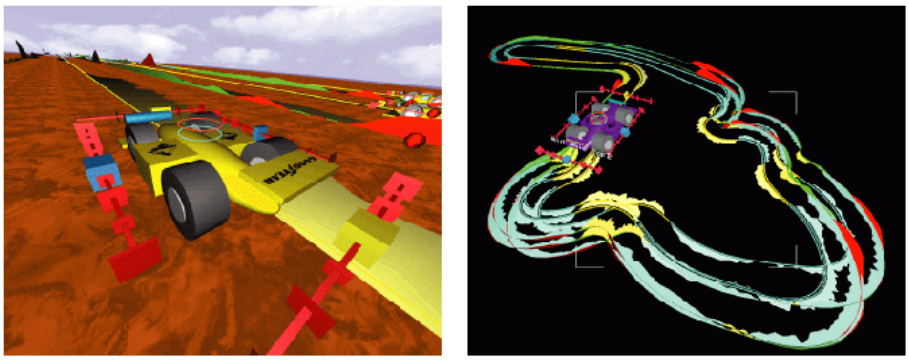


FIGURE 4.11: Visualization of racing car information. Courtesy of Muse technologies.

The model underlying each OpenViz application is the *OpenViz Application Pipeline*. This means that an OpenViz application can be represented as a visualization pipeline consisting of the following steps: 1) access the data, 2) prepare the data and 3) visualize the data. Finally, a user can interact with the visualization through the viewer component. These transformation steps are exactly the same as the steps defined in the visualization reference model. However, interaction that is immediately supported by OpenViz is limited to the viewer component. Therefore, OpenViz is more constrained with respect to interaction than the visualization reference model.

MUSE The *Multi-dimensional User-oriented Synthetic Environment* (MUSE) is a high-end visualization platform that enables the development of multi-user, multi-platform, multi-sensory software applications. It consists of an extensive C/C++ library and a collection of tools to develop custom applications. Muse has been deployed in high-end visualization projects within a broad spectrum of domains including aerospace, automotive, manufacturing, medical, and oil & gas.

Moltenbrey (1999) describes the usage of Muse to visualize data to analyze the performance of tires on racing cars. Figure 4.11 contains two visualizations that help analyzing the tires' behavior. The following quote from that article briefly illustrates the usage of a Muse-based visualization to improve the analysis.

> *Using a multi-sensory interface from MUSE Technologies (Albuquerque, NM) that gathers and displays visual, tactical, and audio data, Goodyear race-tire engineers at the company's Akron, Ohio, Technical Center are able to simultaneously view in 4D (3D images along with a time element) graphical format up to 20 types of data describing the state of the car, such as its acceleration, collected from test runs at actual race*

tracks. The group experiences a lap-by-lap replay of the data, with information about the car's state mapped onto the tires, dials, and vehicle's body, providing a real-time visual representation of performance variables and their effects on the car and tires. By easily visualizing so much data, the engineers have been able to expand their analysis capability, which will lead to improved performance of tires and vehicles in an array of race conditions–and, ultimately, an improved tire design for the consumer market. (Moltenbrey 1999)

4.2.4 Research projects

The fourth and last group comprises research projects instead of (commercial) products. In a research project the main focus is on a particular aspect of visualization systems, such as the distributed software architecture or the user-interface aspects concerning interaction with data sources. To illustrate those aspects, all projects discussed here have associated prototype implementations. The surveyed research projects are the Computational Steering Environment (CSE), Carnegie Mellon University's Visage and Nasa's Visage 3.0[2].

CSE The goal of the Computational Steering Environment (van Liere, Harkes & de Leeuw 1998) was to integrate simulation and visualization in the domain of scientific visualization. Traditional visualization approaches do not visualize any results until the complete simulation has finished. This implies that if users want to experiment with the settings of the simulation, they have to run the complete simulation a couple of times. Especially for complex scientific processes, this may be a time-consuming task.

The goal of the CSE project was to integrate the simulation and visualization loop in such a way that intermediate results are immediately visualized. This enables the scientist to interact with the simulation during visualization (*computational steering*).

To offer software developers the means to build applications supporting computational steering, the software architecture of CSE comprises a distributed blackboard. Conceptually, information producing and consuming components (called satellites in CSE) are all connected to a single blackboard. Using this blackboard, the satellites can exchange all information that needs to be shared.

To increase the performance and scalability of the system, the blackboard has been split up over several distributed blackboards which reside at different machines. Updates at one distributed blackboard are then propagated towards the other local blackboards which have satellites using the same piece of information.

[2]Please note that two projects called Visage exist. One project runs at Carnegie Mellon University, whereas the other one is performed by Nasa. To avoid confusion we will call the Visage projects CMU's Visage and Nasa's Visage 3.0 respectively.

CMU's Visage An elegant implementation of direct manipulation as a means of interaction and visual exploration is achieved by CMU's Visage, which is described in (Derthick et al. 1997) and (Kolojejchick et al. 1997). Visage is *information centric* which means that the information itself is the interface and can, therefore, be manipulated or dropped onto different visualization primitives.

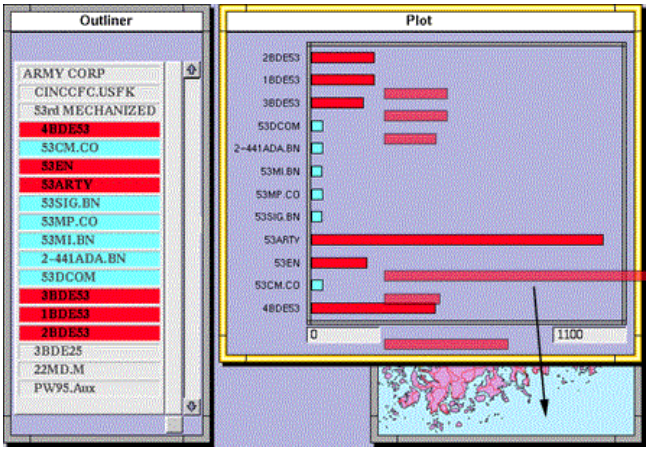


FIGURE 4.12: Data elements in CMU's Visage can be dragged onto different visualization primitives. Courtesy of Carnegie Mellon University.

As an example, Figure 4.12 shows how a user has selected certain elements in the *outliner* on the left hand side of the figure which she has dropped onto the bar chart (*plot*). The result of this interaction is that the plot now only visualizes the selected elements instead of the complete dataset. Interesting elements, for example the longest bar, can subsequently be dropped onto the map for further inspection, e.g. to reveal its geographical position.

Nasa's Visage 3.0 Nasa's Visual Analysis Graphical Environment (Visage 3.0) (Nasa Goddard Space Flight Center 2000) is a distributed visualization development environment written in Java. Visage offers a rich set of Application Programming Interfaces (APIs) to facilitate everything in the visualization process, ranging from dataset manipulation to 2D and 3D viewers.

The visualization architecture of Nasa's Visage consists of three distinct layers: the *DataSet*, the *VisModel* and the *Visualization*. The *DataSet* is the large set of indexed data that a visualization has access to (the datasource). The *DataSet* contains all the information that is available. The mediate-model, the *VisModel* (visualization model), is a window into the *DataSet*. The *VisModel* contains all information necessary to create a visualization. Usually, this is a subset of the *DataSet*. Finally, the *Visualization* represents all information available in the *VisModel*. Additionally, the *Visualization* layer includes a *Manipulator* that contains controls for the user to alter the *VisModel*'s bounds.

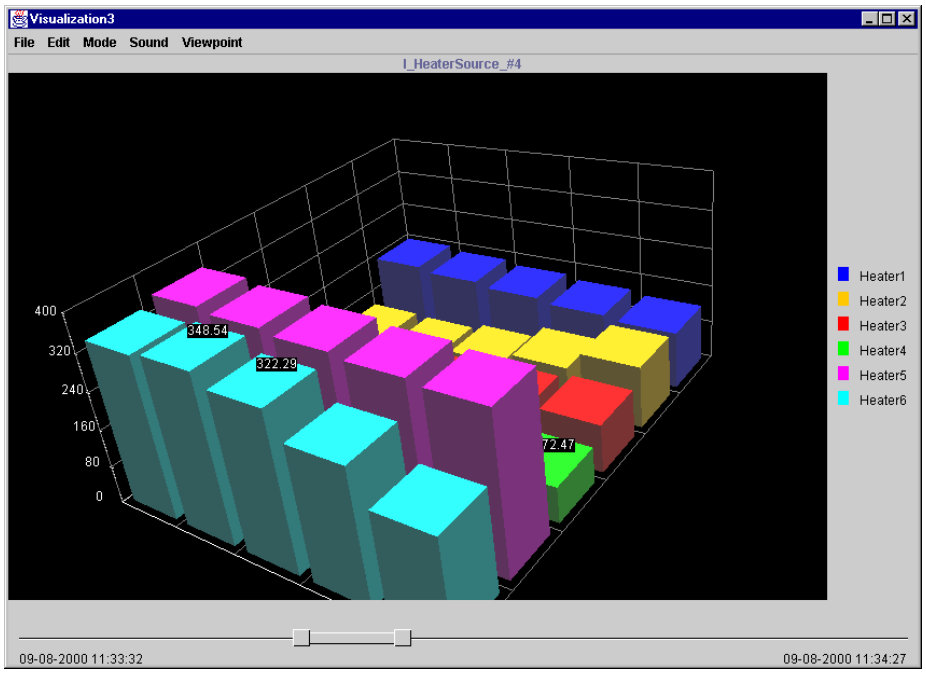


FIGURE 4.13: Nasa's Visage supports dynamic data feeds to visualize dynamic data. Visualizing a time fragment is supported through time-sliders (in the bottom of the window).

A unique property of Visage 3.0 is its support to visualize dynamic data in a distributed environment. To achieve this, a Visage session consists of a server and a couple of clients. The server offers the concept of a *DataFeed* to which a client can subscribe. Consequently, changes in the data are notified to the interested clients which can then update their visualizations. Figure 4.13 shows a screenshot of a Visage client which visualizes dynamic data using a 3D bar chart. Whenever a single value at the server changes, the client is notified, retrieves the changed data and accordingly updates the height of the corresponding bar.

Discussion: data sources – past, present or future The fact that the data coming from the server changes over time implies that the data source is dynamic data (in contrast to static data sources such as databases). In addition to being static or dynamic data sources can concern past, present or future. These data sources have rather distinct characteristics. As a consequence, they require different treatments in visualization applications. Data about the past is static and usually covers a long time period. It is often used for overviews of the complete time period or fragments thereof. Data about the present, e.g. coming from measuring devices, consists of up-to-date information which might

change at any moment in time. Therefore, data about the present only reflects a certain point in time instead of a time span such as is the case with data about the past.

Data about the future is an extrapolation of data from past and present according to some prediction model. Like data about the past, it spans a certain time period. It differs, however, from the past due to the possibility of multiple futures. Namely, future data depends on the deployed prediction model and its current parameter settings which might be changed (interactively) by the user.

Nasa's Visage provides support to visualize (parts of) the past by introducing time sliders (the slider at the bottom of Figure 4.13). As time progresses, the currently selected time frame represented by the time sliders moves to the right. However, users can interact with the progress of time by dragging the bar to another point in time. Additionally, the selected time frame can be adapted by grabbing the left or right end of the time slider and change the size of the slider. As a result of this interaction, the *VisModel* window into the *DataSet* is modified and consequently a different time frame is visualized.

4.3 Summary and Conclusions

To gain a better understanding of the process of visualization, this chapter discussed visualization models from a theoretical as well as a practical perspective. A common denominator of the theoretical models is their description of the process of visualization as a series of transformation steps. They differ, however, in the interpretation of these steps and whether there is support for interaction during the transformation processes.

The visual taxonomy determines two structures, symbolic and visual. Visualization itself is defined as the transition from the symbolic domain into the visual domain. The visualization pipeline focusses on the transformation from source data to the displayed visual representation. It defines filters as the processors which are connected by means of pipes. Support for human interaction is introduced by the visualization reference model. It defines interaction possibilities during data transformations, visual mappings and view transformations. Finally, a formal approach to visualizations makes it possible to prove characteristics of the visualization, and in particular its effectiveness.

To cover visualization models in practice, this chapter distinguished between four types of visualization environments. First, we discussed embedded visualization as employed by Microsoft Excel and Arena. We saw that the tight coupling of data and visualization can lead to a powerful tool, but does not allow us to combine it with other data or visualization tools. Second, we discussed the general-purpose visualization environments AVS Express and OpenDX. Both environments contain a visual programming tool that is based upon the visualization pipeline.

Visualization component libraries such as VTK, OpenViz and Muse provide developers with a rich set of analysis, data manipulation and visualization components. The prescribed visualization model of such libraries is usually based on a variant of the visualization pipeline. However, visualization application developers can just as well deploy the components independently in their own architecture. As a last group, this chapter discussed visualization research projects, each with its own research focus. CSE's goal is to integrate simulation and visualization into a computational steering environment. CMU's Visage illustrates how direct manipulation can be achieved in an information centric visualization. Finally, Nasa's Visage elaborates dynamic data sources in a distributed environment. Additionally, it enables users to conveniently browse through time by means of time sliders.

As a conclusion, we see that theoretical and practical visualization models are closely related. Moreover, most visualization tools are directly based upon a theoretical visualization model. The consequence of this close relation is that whenever the theoretical model has a limit, e.g. a single view on the data, this limit is inherited by the visualization product. Thus, before developing a software architecture for multi-user visualization we first have to take a closer look at a visualization model that supports the multi-user, multi-perspective requirements. The next chapter, therefore, discusses the DIVA architecture on a conceptual level first.

CHAPTER 5

Diva: Distributed Visualization Architecture

Abe said something interesting. He said that because everyone's so poor these days, the '90s will be a decade with no architectural legacy or style – everyone's too poor to put up new buildings. He said that code is the architecture of the '90s.
Douglas Coupland (Microserfs)

Taking existing applications, built and designed for single user purposes, into the multi-user realm is a complicated matter. The underlying architecture of the system is (probably) not designed with multi-user consequences in mind. Adapting the application, therefore, involves changes in the *structure* of the system and requires new forms of communication. Unfortunately, these kinds of modifications are more radical and expensive than source code changes which do not alter the software architecture.

In the field of information visualization, the same argument is valid. Extending a computer program that visualizes a local database with multi-user functions significantly changes the architectural structure of the program. Moreover, it is doubtful whether we should consider it as an extended version of the same application. It is probably more correct to state that we created a new application (a new structure) by reusing parts (functional units) of the original application.

Structure After Section 5.1 briefly describes the context of the *Distributed Visualization Architecture*, Section 5.2 discusses some issues of multi-user visualization. The DIVA architecture is subsequently discussed according to three perspectives: the conceptual architecture (in Section 5.3), the software architecture (Sections 5.4, and 5.5) and, as a last perspective, we look at the information architecture of DIVA in Section 5.6. After a brief recapitulation of the architecture in Section 5.7, a more theoretical discussion of software architecture and its usage is discussed in Section 5.8. Finally, Section 5.9 summarizes and concludes this chapter.

5.1 Distributed Visualization Architecture

This chapter discusses our architecture for visualization in a distributed multi-user environment. The architecture builds upon two important observations. First, multiple users with different backgrounds have individual information needs and thus require **multiple views** on the information. Second, to allow users to experiment with the information and the visualization, the visualization must be **adaptable**. To meet these requirements, our approach decouples the generation and presentation of information by means of an intermediate model, allowing users to adapt the visualization to their information needs.

The **Distributed Visualization Architecture (**DIVA**)** is described by means of three architectural perspectives. The first view comprises a **conceptual architecture**. It provides a high-level overview of the organization of data and functionality of the system. The conceptual architecture is comparable with the (theoretical) visualization models of the previous chapter. A second view is the DIVA **software architecture**. It describes the structure of the system in terms of software components, interfaces and communication. A last perspective is the **information architecture**. This view describes the structure and transformations of the data from source to visual representation.

Together with the two following chapters, this chapter forms a cluster describing information and business visualization from an architectural perspective. The focus of this chapter is on the requirements and the high-level overview of the DIVA architecture. The next chapters illustrate the DIVA architecture from a more practical perspective. Chapter 6 contains a description of several prototype implementations we have created to realize the architecture. Additionally, it discusses the consequences of deploying visualization in computer-supported cooperative work. Chapter 7 describes the usage of the DIVA architecture in combination with a reusable collection of 3D gadgets to visualize business process simulations.

5.2 Multi-user Visualization

Before discussing our architecture for multi-user visualization, we first describe an example application of distributed multi-user visualization. Since the management information case study in Chapter 3 already provides an example of business visualization, this application focuses on the multi-user and collaborative aspects of visualization.

Imagine an international company with offices in countries all over the world. The CEOs of the company have decided that the offices in different countries have to standardize the work process to improve the service level of the whole company. A business process re-engineering (BPR) project is started which must result in standardized business processes based on the different existing ones. The managers in the different countries will make the final decisions about the new work processes at a conference. However, they want to prepare and discuss some alternatives before the actual decisions are made.

We want to deploy information technology to support the managers in studying the alternatives and making the decisions. In addition to Web-pages and email to exchange information, we create business process simulations to *execute* the redesign alternatives (Eliëns, Niessink, Schönhage, van Ossenbruggen & Nash 1996). To fully exploit the potential of the business simulations we want to allow the managers to discuss both the results of the simulation, e.g. the costs and profits, and the running simulation itself, e.g. to illustrate the activities in the redesigned alternative.

This example requires two forms of collaboration: **synchronous distributed** and **face to face** (Ellis, Gibbs & Rein 1991) and (Shneiderman 1998). First, to help the managers prepare for the conference, we require support for *synchronous distributed* collaboration where the users cooperate at the same time but at different places. Second, at the conference, where the decisions are made, the managers discuss the selected alternatives *face to face*, i.e. same time, same place.

By taking a closer look at the example, we can discriminate some roles that the participants of the distributed visualization play. First, there is the role of the **talker** who demonstrates a redesign alternative to a number of **listeners**. The listeners follow the explanation and can browse through the resulting information. They are not allowed to interact with the business simulation itself. However, to increase the interactivity of the session, a number of people may be allowed to interact with the simulation. We will call them the **interactors**.

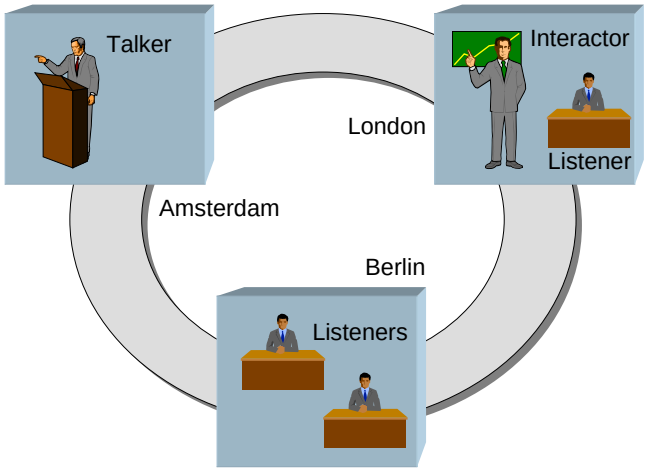


FIGURE 5.1: Example configuration of a multi-user visualization system

Figure 5.1 illustrates an example configuration consisting of one talker, one interactor and three listeners. The talker in Amsterdam presents the results of the running simulation to the interactor in London and the three listeners (two in Berlin and one in London). In the remainder of this chapter, we will show how the information coming from the talker can be distributed to the interactors and listeners. Additionally, we will discuss what software architecture is needed to support this process.

5.3 Conceptual Architecture

As we have seen before, the process of visualization transforms raw data into visual representations. In a single user/single machine environment, a visualization program can read and process the data on the harddisk and display the visualization on the monitor. When we extend this to a distributed multi-user environment, things get more complicated and we have to rethink the architecture.

As an introduction to the DIVA architecture, this section discusses the conceptual architecture. It gives an overview of the process of visualization comparable with the visualization models discussed in Chapter 4. However, our conceptual architecture extends those models with multi-user and collaborative aspects.

5.3.1 Primary, derived and presentation model

We regard the process of visualization as a transition of data through a sequence of models, starting with the generation of data and ending with the presentation of a visualization. To allow for multiple, possibly shared, perspectives on the data, we introduce an intermediate model between the generation and presentation of information. This intermediate model contains information based on the originally generated data, adapted to the information needs of its users.

Figure 5.2 depicts our architecture on a conceptual level. It contains three models going from symbolic data in the primary model to a visual representation in the presentation model.

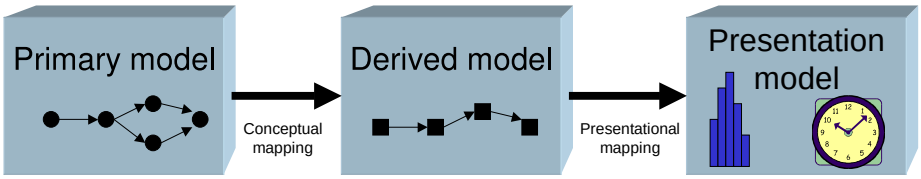


FIGURE 5.2: Conceptual architecture

The **primary model** is the source of the visualization. It contains explicitly or implicitly all information that is available. When the source data is static, the primary model may, for example, be a database. In this case, the primary model contains explicitly all data the system is using to make a visualization.

In contrast, when simulation is used to generate dynamic data, the primary model is the simulation model executed by the simulation application. The raw data generated by the simulation usually is only of minor interest. Derived results, such as average waiting times and standard deviations, are better suitable as source data for visualization. These derived values are stored in the next model.

The intermediate model between the generation and presentation of information is the **derived model**. Since the derived model has adapted the data from the primary model to the information requirements of its users, it is used as the information source for the presentation model. Multiple derived models, based on a single primary model, can be created to serve multiple users.

The rightmost box in Figure 5.2 is the **presentation model**, which is used to display the visualization. The contents of the presentation model depend on the presentation technique used, e.g. for 3D visualizations the presentation model is the 3D scene graph. Because the derived and presentation model are decoupled, users can share the derived model while their presentation is different. The information contents of the visualization is the same although the *style* can differ depending on the available platform and the user's preferences.

5.3.2 Transition from model to model

In addition to the three models described above, Figure 5.2 contains two transitions or mappings between the models. The **conceptual mapping** from the primary to the derived model gives us the flexibility to adapt the data space to our information needs. This is useful because we are only interested in information with some value for our current interest; data only becomes information when it is of use to answer the questions we have. As a consequence, data in the derived model differs from data in the primary model in two ways. First, only data that is useful for the current perspective is selected for the derived model. Second, information derived from primary data is added to the derived model.

The mapping from the derived to the presentation model, the **presentational mapping**, specifies how information in the derived model is presented. Data elements in the derived model are mapped onto visualization primitives. What information is selected and how it is represented is up to the users of the system, their information requirements and their personal taste. For example, two different presentational mappings can be deployed to present the same information in a 2D or 3D visualization.

5.3.3 Example configuration of the conceptual model

To illustrate the conceptual architecture, Figure 5.3 contains an example of the Amsterdam-London-Berlin architecture consisting of one primary model (the business simulation), two derived models and the five persons (views) in the three different cities. The arrows illustrate the flow of data and information through the system. For simplicity, the flow of control is not drawn in the figure.

As can be seen from the figure, four out of five users share the same derived model. They share the same information that is used for the visualization. However, although the information content is the same, the four listeners can display the information according to their preferred style. The listener in London may use moving objects to illustrate activity whereas the talker in Amsterdam employs color to visualize the same information.

One person in Berlin focuses on a different aspect of the simulation and, therefore, has her own derived model. She might, for example, be interested in the resource allocation aspects of the business process simulation. She requires distinct information which is gathered in *Derived model II*.

5.3.4 Relation to other visualization models

The DIVA conceptual architecture is closely related to the theoretical models described in Chapter 4. Most models are based on a process of transforming

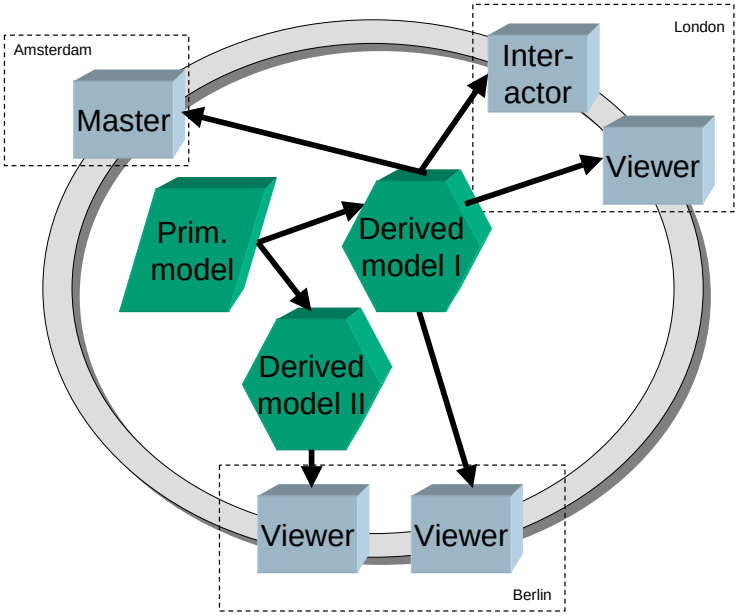


FIGURE 5.3: Example of the conceptual architecture

data through intermediate models. The main difference, however, between our approach and the visualization pipeline or visualization reference model is the possibility of multiple derived models. Our conceptual architecture supports multiple visualization perspectives by means of multiple derived models (as shown in Figure 5.3). Summarizing, our approach can be seen as a multi-user, multi-perspective extension of previously discussed visualization models.

5.4 Basic Software Architecture

A software architecture can be thought of as an artefact that shows how functional and quality requirements of a system can be met. Thus, before we present the DIVA software architecture, we first briefly discuss the underlying requirements.

To structure the discussion, we will introduce the DIVA software architecture in two phases: the basic software architecture and, in the next section, extensions to this basic architecture. This way, we show how extra requirements influence the basic architecture. Moreover, since the addition of expected extensions to the software architecture is evaluated, we, consequently, probe the flexibility of the basic architecture.

5.4.1 Basic requirements

Below, the three main requirements for the DIVA software architecture are given. The requirements are based upon the research problems that we have described in Chapter 1, which are 1) the lack of multi-user support, 2) tight-coupling between data and visualization and 3) limited interaction with the process of visualization.

Additionally, the requirements are influenced by the results of case studies with distributed information visualization. Especially, the evaluation of the Gak management information project as described in Chapter 3 played an important role in giving shape to the requirements and their reflection in the basic architecture.

Basic Requirements

1. **Support for multiple users.**
2. **Support for multiple perspectives.**
3. **Decoupling of data source and visualization.**

The first requirement, the support for multiple users, directly reflects the idea we had at the beginning of the DIVA project. Current software support is too much aimed at the single user whereas visualization is often used in a multi-user context, e.g. decision-making. The DIVA project must provide a means for multiple users to share their visualizations in order to achieve better results from their cooperation.

A direct consequence of our wish to support multiple users is the requirement of supporting multiple perspectives. As becomes clear in the application of visualization in practice, people use information differently. For example in a business setting, financial and operational managers are interested in the same data, but at a different level. The financial manager needs a financial overview, whereas the operational manager requires insight in abnormal financial transactions. Hence, multiple perspectives on a shared information source are required.

The third basic requirement is also directly related to one of the problems we try to solve in this research, namely tight coupling. Instead of tightly integrating specific information sources with specific visualization facilities, we propose a model that decouples source from visualization. This way, information resources can be reused by multiple visualizations while at the same time, visualizations can be reused to present different information sources.

5.4.2 Basic software architecture

This section presents the basic software architecture of DIVA (see Section 5.8 for a more theoretical background on software architecture). The DIVA software architecture reflects the three requirements specified above. Additional requirements demand extensions of this basic architecture. What consequences these extended requirements have on the basic architecture is the topic of the next section.

To describe the DIVA software architecture, we both use simple charts and the **Unified Modeling Language (UML)**. A specification in UML consists of a collection of diagrams, including use-case diagrams, class diagrams and collaboration diagrams. A short introduction into UML can be found in (Fowler 1997). More detailed information about UML is available in (Rumbaugh, Jacobson & Booch 1999) and (D'Souza & Wills 1999).

The choice to use UML instead of an **Architectural Description Language (ADL)** is twofold. First, UML is becoming the *de facto* standard for modeling systems at different levels, including the architectural level. Second, ADLs are currently still in the phase of research and do not offer much advantage for the purpose of clarifying the DIVA software architecture. By sticking to a well known modeling language, it will hopefully be easier to grasp what is really important: the *contents* of the architectural description.

The keyword of the Distributed Visualization Architecture is **decoupling**. Data generation, storage and presentation are not combined into in a single entity (component) but are considered as separate parts contributing to the overal process of visualization. We therefore distinguish between the role of **data provider** and the role of **data visualizer**. The exchange of information from a data provider to a data visualizer takes place via a communication mechanism that we will call the **communication space**.

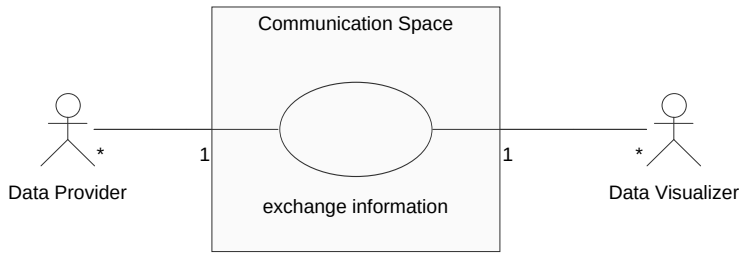


FIGURE 5.4: The basic architecture decouples data provider and data visualizer. Information exchange from the provider to the visualizer takes place via a communication space.

Figure 5.4 gives a high-level overview of the DIVA software architecture. The basic architecture consist of a (logically) single communication space, a number

of data providers and a number data visualizers. The figure shows that multiple data providers, such as databases and simulations can be used as a source for multiple visualizations. However, all information exchange goes through a single communication space. In the remainder of this section, we will show how this basic architecture satisfies the three basic requirements.

Multi-user support (1)

DIVA is a **distributed** architecture: data sources and visualizers can be running on different machines. Furthermore, multiple visualizers can be using the same information source as input for their visualizations. As such, multiple users can visualize a shared information provider, for example they can look at a common database or use the same simulation.

In the DIVA architecture, the mediating communication space is responsible for distributing the information to the visualization components. As a consequence, the information provider is released from the task of maintaining to whom it should distribute its data. At the same time, the visualizer does not have to keep up with multiple information sources. The information provider publishes all it knows to the communication space whereas the visualizer can find all the available information at the same location.

Multi-perspective support (2)

Closely related to the requirement of multi-user support is the requirement for multiple perspectives. Different users have different backgrounds, different tasks to accomplish and hence different information needs. Consequently, the DIVA software architecture must support multiple perspectives.

To achieve this, the software architecture must enable us to build a system like the one specified in Figure 5.1 consisting of a single data source, multiple derived models and multiple views on the derived models. To show how such a visualization session can be realized using DIVA we need to take a closer look at the communication space.

In DIVA the mediating communication space is the **Shared Concept Space (SCS)**. The Shared Concept Space, which will be discussed in detail in Chapter 8, is a model for (distributed) data exchange and is based on the talker-listener pattern. The SCS consists of hierarchical **concepts** that represent the actual data as well as a meta-level description.

The role of the Shared Concept Space in the software architecture of DIVA is illustrated in Figure 5.5. The Shared Concept Space, shown in the middle of the figure, contains two trees of concepts both representing a different derived model based on the source data coming from the data provider. Listeners can present different perspectives on the original data by visualizing (a part of) one of the offered derived models.

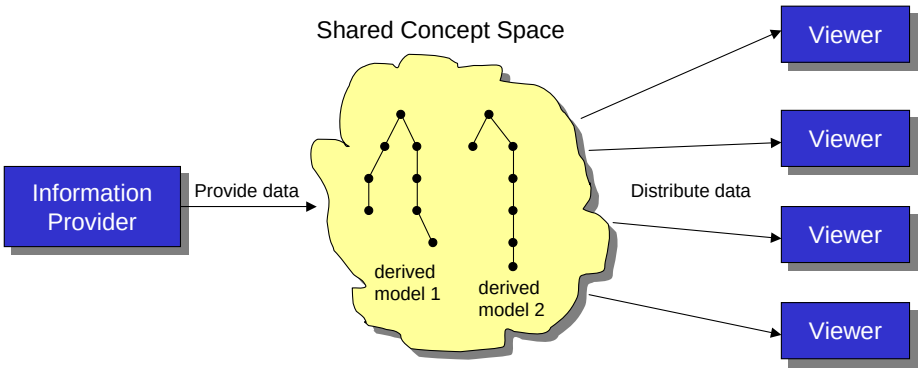


FIGURE 5.5: The Shared Concept Space allows for multiple derived models and hence multiple perspectives on a single information source.

Decoupling (3)

After all that has been said about the basic software architecture, the last requirement, decoupling information provider from visualization, is almost trivial. Moreover, the decoupling of information provider and visualizer is exploited to satisfy the multi-user, multi-perspective requirements.

5.5 Extending the Software Architecture

This section shows how the basic software architecture of DIVA can be extended to support additional requirements. We use the same structure as employed in the previous section. Hence, we first briefly give and motivate the additional requirements. After that, we show how to adapt or extend the basic architecture to satisfy those additional requirements.

5.5.1 Extended requirements

The basic software architecture is capable of providing multiple users with multiple perspectives in a decoupled architecture. The requirements presented below are not necessary for this basic visualization functionality. Nevertheless, they are desirable in information and business visualization applications as we have demonstrated in case studies such as described in Chapter 3.

Extended Requirements

4. **Decoupled data manipulation.**
5. **Interaction with data source.**
6. **Collaboration support.**

The first additional requirement directly comes from our experiences with the case studies at Gak Netherlands. In those studies we found that data available in data sources does not necessarily have the right level of abstraction to be used for visualization. Data manipulation is used to derive 'new' information from data available in the data source. By introducing data manipulation in one of the components of the architecture instead of relying on manipulation facilities in the data source or visualization primitives, transformation rules can be reused for multiple sources and visualizations. For example in the case study at Gak, we specified a derived property that calculated the number of benefit applications at each stage in the business process based on information from the database. This piece of knowledge could later on be reused to derive the same information in the case we used simulation as the data generator.

The second extra requirement concerns the support for better interaction with the visualization process. Presenting information visually improves the insight, however, interacting with the source data and getting feedback increases understanding. The usefulness of interaction is very well expressed by the following Chinese proverb:

> *What I hear, I forget;*
> *what I see, I remember;*
> *what I do, I understand.*

A final additional requirement is support for collaborative visualization. Usually, visualization is a single user activity. However, visualizations are an excellent means of communicating a vision. Visualizations are therefore often deployed to explain statements or convince colleagues. In an interactive visualization system, only sharing the resulting image of the visualization is not enough. More than that, a collaborative visualization system must explicitly support cooperation in the form of perspective sharing and direct communication between interested parties.

5.5.2 Extensions to the basic software architecture

The DIVA software architecture as it has been discussed up to now meets the basic requirements. To satisfy the additional requirements of decoupled data manipulation, interaction and collaboration, the basic software architecture has to be extended. This section discusses how each of these additional requirements influences the basic architecture.

Decoupled data manipulation (4)

To reuse derivation knowledge independently of information provider and visualizer, it has to be decoupled from them. In DIVA, the Shared Concept Space (SCS) is the connection between the data provider and visualizer components. It is the means of decoupling data provider and consumer. The SCS is therefore the right place to enrich the available data with derived information.

In order to be able to derive new information, the Shared Concept Space contains, in addition to the data concepts, **derived concepts**. Derived concepts are produced by separate **processor** objects that listen to available information, process it and produce new derived information.

The usage of derived concepts and processor objects is illustrated in Figure 5.6. Data coming from a data provider is distributed by the SCS to all interested visualizers (this is part of the basic architecture). To support decoupled data manipulation, the available information is also distributed to the processor objects. Based on the received data, the processor objects now publish *derived data* that, subsequently, is distributed by the shared concept space.

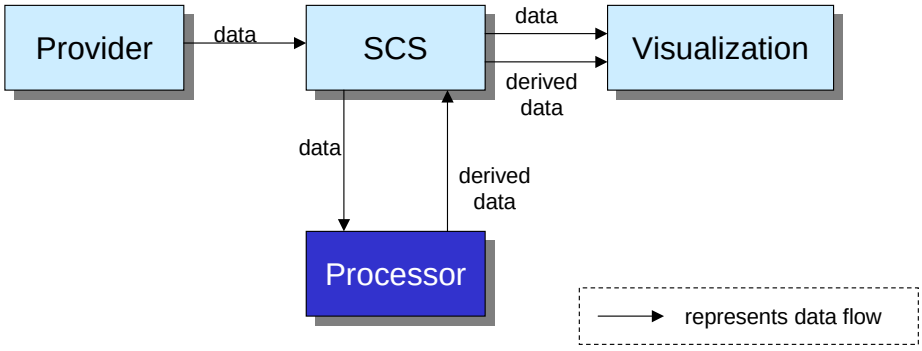


FIGURE 5.6: The flow of data from data provider to simulation is enriched with derived information that is produced by separate processor objects.

As a simple example, imagine a business process simulation that produces 'waiting times' at a particular queue. Whenever the simulation publishes a new waiting time, the processor object adds that time to its internal sum, divides that by the number of produced times and publishes the average waiting time as derived information.

Interaction (5)

The second extension to the basic DIVA software architecture concerns support for interaction. Our goal of creating a fully interactive visualization architecture means that we must be able to interact with every aspect of the visualiza-

tion process. Therefore, we now show how to extend the architecture with a means to interact with the complete visualization process.

First, we discuss how to interact with the data visualizer component. Until now, we described the data visualizer as a single component. For the purpose of interaction and collaboration, however, the component must be split into two parts.

The first element of the visualizer component listens to information in the Shared Concept Space and transforms that into visual structures. For reasons that will be explained later, this element is called a **display agent**. For example, when a vector of integer numbers must be visualized as a bar graph, the display agent maps the integer values to a structure describing the contents of the bar chart, including width, height, color and shape of the bars.

These visual structures are input for the **viewer** component which combines multiple visual structures into a single image, animation or 3D scene. Additionally, the viewer can apply visual transformations, such as zooming, color effects and multiple viewpoints.

Figure 5.7 shows the DIVA software architecture that is enhanced with four levels of interaction support. Every component containing behavior that can be interacted with exposes an interface that can be used for interaction with that component.

The **view control interface** allows for view transformations such as changing viewpoint or modifying the deployed display methods. The offered methods and interaction possibilities depend on the deployed viewer. Figure 5.7 contains an example interface for a 3D viewer.

The mapping from information to a visual structure in the display agents can be influenced via the **display agent control interface**. Since each type of agent performs a specific mapping, the contents of the control interface is dependent on the deployed display agent.

While controlling existing display agents is a means to adapt existing visualizations, new visualizations can be generated by adding display agents to the system. Together, these options form DIVA's possibilities to interact with the visual mapping.

The third level of interaction comprises the manipulation of the data processor objects. In the DIVA software architecture, data manipulation is comparable with the visual mapping. Users can modify the deployed data manipulation by means of adding/removing processor objects or by modifying the settings of the running processor via its control interface.

Finally, interaction with the data generation is achieved via a separate control interface. For example to control a running simulation, a user can start, stop and reset the simulation. Additionally, to experiment with different settings of the simulation, the interface can contain methods to change important parameters of the simulation model.

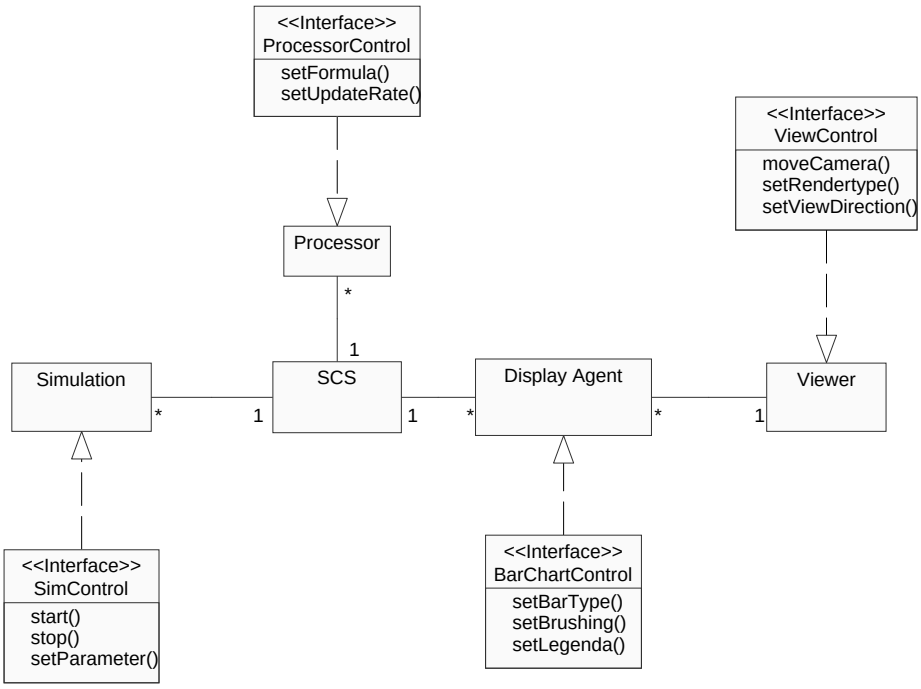


FIGURE 5.7: The Diva software architecture is enhanced with four levels of user interaction.

Collaborative visualization (6)

The sharing of data sources by multiple users is a first step towards collaborative visualization. However, issues such as collaborative sessions, perspective sharing and direct user-to-user communication also play an important role in collaborative visualization. Incorporating them into the DIVA software architecture is not straight-forward and requires further study. The results of such a study are discussed in Chapter 6.

5.6 Information Architecture

After the conceptual and software view on DIVA, the third perspective on DIVA is a view on the underlying information architecture. In a visualization system, at least something about the structure of information has to be known since data providers and visualizer have to 'understand' each other.

Data sources can contain data in every possible format, from completely unstructured to structured tables including meta-descriptions. Additionally, data visualizers can require information in different input formats. It is impossible

for the mediating concept space to understand all those data structures. Therefore, the Shared Concept Space must provide a standard way of describing data and structure, the information architecture. The prescribed format of the SCS is a hierarchy of typed concepts.

5.6.1 Hierarchical data and derived concepts

In DIVA, the Shared Concept Space consists of hierarchically organized data elements (the concepts) of a particular type and structure. Each **data concept** contains elements of a basic type, such as integer, real value, string, in a determined cardinality. Thus, a single concept can contain a single integer, or an array of strings, or a 2-dimensional matrix of real values, etcetera.

The combination of basic data type and cardinality allows data provider and visualizer to exchange any type of data. By organizing those data elements in a hierarchy, both provider and visualizer can structure their information. For example, each concept can consist of two subconcepts, one containing the actual data, and a second one containing a description of the data. Another possibility is the ability to group coherent data in a single concept hereby making it easier for the visualizer to find interesting data.

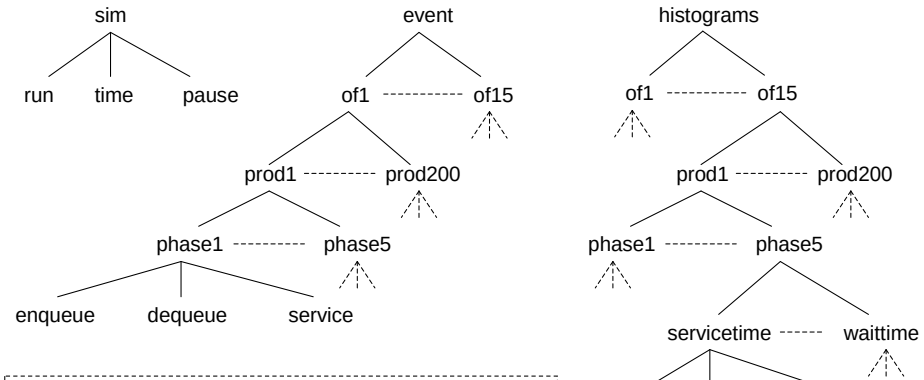
From the visualizer's point of view, **derived concepts** are similar to data concepts. However, they are not provided by the data source, but produced by processor objects. Derived concepts are integrated into the hierarchy of concepts just like the data concepts.

5.6.2 Example of the information structure

Probably the best way to explain the information structure of DIVA is through an example. Therefore, we show a fragment of the Shared Concept Space as it was defined in the case study of Chapter 3. A schematic overview of the example information architecture is given in Figure 5.8.

Three main concepts split the data elements in three different groups, one for general information about the simulation, one for information about the running simulation and one group containing derived statistical information. The `sim` concept contains general information about the simulation such as whether the simulation is running or not and what the current simulation time (`sim.time`) is.

The `event` concept contains all information about the simulation events as they occur in the discrete event simulation of the business process. They are structured according to office, product and product phase. For example when a new entity with ID 15 is being served at stage 3 of product AW008 in office 12, the concept `event.of12.aw008.phase3.service` is set to the integer value of 15.



sim	
- time (int)	simulation time
- run (boolean)	is sim running
- pause (boolean)	is simulation paused
event	
- <of>	office
- <product>	product type
- <phase>	phase within business process
- enqueue (int)	new entity arrived at waiting queue of phase
- dequeue (int)	entity left waiting queue of phase
- service (int)	entity is served at specified phase, product, office
histograms	
- <of>	office
- <product>	product type
-<phase>	
- <type>	type of histogram
- description	description of histogram
- <week> (double[])	data containing the histogram for specified week

FIGURE 5.8: A fragment of the information structure of the Asz/Gak case study incorporating simulation and derived statistical data.

Whereas `sim` and `event` are data concepts supplied by the simulation, the `histograms` concepts are derived properties provided by separate processor objects. They contain statistical information based on data coming from the simulation. Like the `event` concepts, they are organized according to office, product and phase. Since multiple histograms are available for each phase a `<type>` subconcept discriminates the available histograms.

The heights of the bars in the histograms are represented by a sequence of numbers where each number represents the height of a single bar. The responsible processor object creates a new histogram at the end of each week in `histograms.<of>.<product>.<phase>.<type>.<weeknr>`. More information about the histogram, such as a description of the contents, is provided in a separate description concept (`histograms.<of>.<product>.<phase>.<type>.description`).

Depending on the task the visualization has to support, the visualizer can use a subset of all information in the shared concept space as input. For example, an up-to-date visualization of the running simulation will listen to all `sim` and `event` concepts, whereas a visualization that gives a historical overview is

probably more interested in the `histograms` part of the concept space.

5.7 Diva Architecture recapitulated

Multi-user visualization allows a group of people to visualize shared information sources in common or individual manners. The distributed visualization architecture DIVA aims at supporting the processes of multi-user visualization by providing an extensible architecture to connect static and dynamic data sources, data manipulation components and a collection of presentation modules.

The key to understanding the DIVA architecture is decoupling. Information sources and data generators are separated from information presentation. This allows for multiple, possibly shared, views or perspectives on the information sources. Additionally, multiple information sources can be combined into one coherent visualization.

This chapter presented DIVA by means of three perspectives on the architecture: the conceptual, software and information architecture. The conceptual architecture describes the process of visualization as a transition of data through a sequence of models. Through a conceptual mapping, it allows for multiple derived models which reflect different information perspectives. Each derived model is transformed into a presentation model which contains the information necessary to display the visualization.

The DIVA conceptual model of visualization is realized by the software architecture. The software architecture is discussed in two parts. The basic architecture meets the requirements of multiple users, multiple perspectives and decoupled data source and visualization. The additional requirements show how the software architecture should be enhanced to become an adaptable, interactive, collaborative visualization architecture.

A critical element of the DIVA architecture is the Shared Concept Space (SCS), a model for (distributed) data exchange. The information architecture defines the structure of the SCS as a hierarchy of concepts. Data concepts are linked to data entries directly coming from the data provider. Derived concepts, on the other hand, add new information based on other data elements. Decoupled derived concepts allow for the reuse of derivation knowledge for multiple data sources and multiple visualization sessions.

5.8 Software Architecture revisited

This chapter shows the distributed visualization architecture according to three points of view. One of these perspectives concerns the software architecture. The software architecture is considered as a high-level design description. And

although the structure of software systems has long been an issue of concern, e.g. (Dijkstra 1968) and (Parnas, Clements & Weiss 1985), software architecture has only recently emerged as an explicit field of study.

5.8.1 Definitions

The idea behind software architecture is the same among most architects and comes down to something like *the gross structure represented as a high-level organization of computational elements and interactions between those elements*. However, every architect seems to have his or her own definition of the term software architecture which includes extra aspects, considered relevant by that particular architect. To illustrate the commonalities and differences between the definitions we will give some of the more well-known definitions below.

> *The structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time (Garlan & Perry 1995).*

The first definition given here was developed at a software architecture discussion at the Software Engineering Institute in 1995. It adheres to the basic idea of structure of the components and the relationships between them. An extra feature added by this definition is the fact that a software architecture includes guidelines governing the current and future design.

As a second perspective Shaw & Garlan (1996) explain that the architecture of a software system first of all defines a system in terms of computational components and interactions between those components. In addition to this, the architecture shows the correspondence between the system requirements and elements of the constructed system. This allows architects to specify and verify important quality requirements such as capacity, scalability, consistency, etcetera.

Third, Buschmann, Meunier, Rohnert, Sommerlad & Stal (1996) define the architecture as the constellation of subsystems and components and the relationships between them. Additionally, they define terms used in the definition such as component, relationship, view, functional and non-functional properties and software design.

> *A software architecture is a description of the subsystems and components of a software system and the relationships between them. Subsystems and components are typically specified in different views to show the relevant functional and non-functional properties of a software system. The software architecture of a system is an artifact. It is the result of the software design activity (Buschmann et al. 1996).*

The definition which is sometimes being referred to as the current standard definition of software architecture is given by Bass, Clements & Kazman (1998). This definition defines an architecture in terms of components, the properties thereof and the relationships among them.

> *The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components and the relationships among them. (Bass et al. 1998).*

Bass et al.'s (1998) definition contains some important implications. The following issues come from (Bass et al. 1998). First, architecture defines components. Since a software architecture only comprises the externally visible properties of components, it abstracts from the internal working of components but provides a high-level overview of elements and interactions. Second, a system can comprise more than one structure, where each structure describes the conception of the architecture from a different perspective. Finally, the behavior of each component is part of the architecture as long as the behavior can be observed. This means that graphs depicting components and *contains* and *uses* relations are not sufficient to be called architecture. At least some notion of the behavior is obligatory according to this definition.

The last important definition of the term software architecture comes from the draft recommended practice for architectural description (P1471/D5.2) prepared by the Architecture Working Group of the Software Engineering Standards Committee.

> *Architecture: the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution (Architecture Working Group 1999).*

This definition differs from other definitions in two ways. First, it avoids the use of the term *structure* because that is often being related to physical structure while a software architecture focusses on concepts, hence *organization*. The second dissimilarity concerns the inclusion of the environment into the definition. An architecture cannot be viewed in isolation but only in the context of its environment.

5.8.2 Use of the term architecture

As can be concluded from the multitude of definitions of software architecture available, the term *architecture* in software is used in different ways. Garlan & Perry (1995) give three common uses of the term.

- The architecture of a particular system, as in "the architecture of this system consists of the following components."
- An architectural style, as in "the system adopts a client-server architecture."
- The general study of architecture, as in "the papers in this journal are about architecture."

The term architecture as used in the acronym DIVA (distributed visualization architecture) is mainly referring to the architecture of a particular system (or group of systems). The DIVA architecture describes a software system that can support multiple users in visualizing information in a distributed environment.

A second perspective on the DIVA software architecture is more in the tradition of architectural styles. The architecture describes an architectural pattern that might be used in domains other than visualization. The architecture described in this chapter may, amongst others, be deployed for multi-user chat systems, or for virtual environments with a focus on information and knowledge.

Instead of reusing the complete architecture, one might also abstract from particular elements of the architecture and reuse them as architectural patterns or styles. Chapter 9 describes architectural styles for information exchange in distributed object-oriented systems, which are based on experiences with designing DIVA and its implementations.

5.8.3 Why is software architecture important?

A relevant question covered by some authors in their introductions to software architecture is why architectures are important? What do we gain by investing time on specifying an architecture before building a software system?

In their introduction to the special issue on software architecture in the IEEE transactions on software engineering, Garlan & Perry (1995) state that *a principled use of software architecture can have a positive impact on at least five aspects of software development*: understanding, reuse, evolution, analysis and management.

As a first contribution discussed by Garlan & Perry (1995), a software architecture can abstract from a large and complex design and can focus on the high-level constraints to improve the **understanding** of the system as a whole. Second, architectural designs increase the possibility of **reuse** of both individual components and the architectural framework. For example, Gamma, Helm, Johnson & Vlissides (1994) and Buschmann et al. (1996) describe an array of reusable patterns. Since an architecture makes the dimensions along which a system is expected to **evolve** explicit, we can better understand the costs and implications of certain changes. Fourth, architectural descriptions provide new

opportunities for **analysis**, such as conformance to an architectural style and conformance to quality attributes. Finally, Garlan & Perry (1995) state that software architecture can have a positive impact on **management** when a software architecture is seen as a key milestone in the software development process. Since an architecture makes development issues, such as functional and non-functional requirements, and expected changes explicit, the risk of creating an inadequate or inflexible system is decreased.

In the light of these reasons why software architectures are important, we will briefly reconsider the expected contribution of the DIVA software architecture. Contribution number one is understanding. By analysis and experimentation, we try to understand the issues that are important in systems supporting multiple users in visualizing shared information. The DIVA software architecture makes these issues explicit. A second purpose is a clarification of the effects of evolution on the architecture: what impact do additional requirements have on the software architecture? To illustrate this, the description starts with a basic architecture and shows how it can be refined or extended to facilitate new requirements.

The architecture discussed in this thesis can be reused in other projects or systems. DIVA constitutes a high-level transferable abstraction which can be reused as a whole or in parts. Candidate systems include other visualization applications or systems which concentrate on the information dissemination among multiple users. Finally, the DIVA architecture and the prototypes based on it have lead to patterns for distributed object-oriented systems which can be deployed in other, possibly related, projects.

5.9 Summary and Conclusions

This chapter discusses the *Distributed Visualization Architecture*. The DIVA architecture is aimed at providing a flexible architecture for multi-user, multi-perspective visualization in a distributed environment. A key aspect of the architecture is the decoupling of information provider and visualizer by means of an intermediate data model (the Shared Concept Space).

In its current form, DIVA is an architecture supporting multiple (possibly dynamic) data sources, and multiple perspectives and users. It allows for decoupled data manipulation and enables multiple levels of interaction with data source and visualization mappings. The next two chapters illustrate the usage of DIVA as a basis for experiments with multi-user and collaborative visualization.

CHAPTER 6

Experiments with Visualization and Collaboration

The fundamental principle of science, the definition almost, is this:
the sole test of the validity of any idea is experiment.
Richard P. Feynman

During the DIVA project, a series of case studies and experiments in the form of prototype implementations has been performed. The experiments turned out to be useful to test and improve the DIVA software and information architecture. Additionally, they provided a means to evaluate extensions of the basic architecture in practice. One of those extensions, collaborative visualization, will be discussed more extensively and is the topic of the latter part of this chapter.

6.1 Overview of Prototypes

Before describing the DIVA experiments in more details, a brief overview of the more important case studies and experiments is given. During the project, four experiments have been performed, each with a different goal, context and technology (Figure 6.1).

Prototype	Description	Architectural aspects	Technology
Modern Times (1997)	Visualize (simple) business process simulations by means of a factory metaphor.	The simulation distributes updates to visualize dynamically changing data. This prototype does not contain a SCS	C++, Corba and The Visualization Toolkit (VTK)
The Great Dictator (1998)	Focuses on collaborative visualization (*so many users — so many perspectives*)	Visualization perspectives are contained in a repository and can be exchanged among the visualization participants	C++, Java, Corba, Voyager and VRML
Gak Management Information (1999)	2D visualization of management information. The data sources comprise both databases and business simulations.	Combination of static and dynamic data. Support for decoupled data manipulation and derived concepts.	Java and Voyager
3D BizViz (1999)	A reusable collection of 3D gadgets is deployed to visualize business process simulations.	Support for decoupled data manipulation and derived concepts.	Java, Voyager and Java3D

FIGURE 6.1: Overview of the prototypes which have been done during the Diva project.

Before the DIVA project started, the idea of, and a technical infrastructure for, integrating (business) simulations with Web-based representations was described by Eliëns et al. (1996). Although this study was not explicitly part of the DIVA project, it provided a starting point for the distributed visualization architecture. Based on some of the notions developed there, we created the first DIVA prototype.

The first DIVA prototype visualizes (simple) business process simulations using a factory metaphor; hence its name *Modern Times* after Charlie Chaplin's movie. The prototype shows how we can visualize dynamically changing information by sending updates to the remote visualizations. Section 6.2 describes the *Modern Times* prototype in somewhat more detail.

The second prototype, *The Great Dictator (TGD)*, mainly focuses on collaborative visualization. Perspectives are modeled as independent entities that can be exchanged between a group of users. Additionally, a repository contains perspectives that can be downloaded to a local machine to visualize information. The data in the Shared Concept Space, of which a light-weight version is available in TGD, is provided by a running simulation. To control the simulation a remote control is shared by the participants of the visualization process. More information about collaborative visualization and the TGD prototype is given in Section 6.3.

After two rather technical explorations, we felt a necessity to do a more ex-

tensive case study in a real world situation. The results of the *Gak management information visualization* project have already been presented in Chapter 3. Since data manipulation plays an important role in presenting information from multiple data sources, support for decoupled data manipulation and derived concepts was the main focus of this project.

The last case study takes the visualizations of the *Gak management information* case study as a starting point and, subsequently, provides a 3D visualization. To achieve this, we introduce a small set of 3D visualization gadgets and associated behaviors that proved to be relatively complete for our case. Chapter 7 describes the 3D case study in more detail and, additionally, illustrates the usability of the gadgets and associated behaviors.

6.2 Modern Times

In *Modern Times*, decision makers deploy business process simulation, the Web and visualization to improve their information supply and, consequently, improve the quality of their decisions. For example, in a business process reengineering project, people involved can study and compare redesign alternatives before deciding on the best option. In the *Modern Times* prototype the improved information supply consists of a combination of Web-pages and visualizations of business simulations.

Figure 6.2 contains a screenshot of *Modern Times* showing the EYE Web-browser (Eliëns et al. 1997) with integrated visualization. The visualization uses a factory metaphor to present the simulation. It uses rotating objects and conveyer belts to visualize activity and flow within the simulation.

The visualization shows a counter-based business process in which clients go to a desk where they are served by a clerk. This clerk does some work for the client during which the clerk requires access to an information archive. The visualization shows the following concepts: boxes representing the clients, a sphere representing the clerk, a cylinder representing the information archive, and two conveyor belts representing the stream of clients and information through the simulation.

6.2.1 Software architecture

The software architecture of *Modern Times* consists of four major components: a Web-server, the simulation, the browser and the visualization plug-in, as illustrated in Figure 6.3. The simulation acts as the primary data generator and transmits its internal simulation data directly to the Web-browser's visualization component.

When compared with the general DIVA software architecture discussed in Chapter 5, the *Modern Times* prototype is only a limited implementation of the

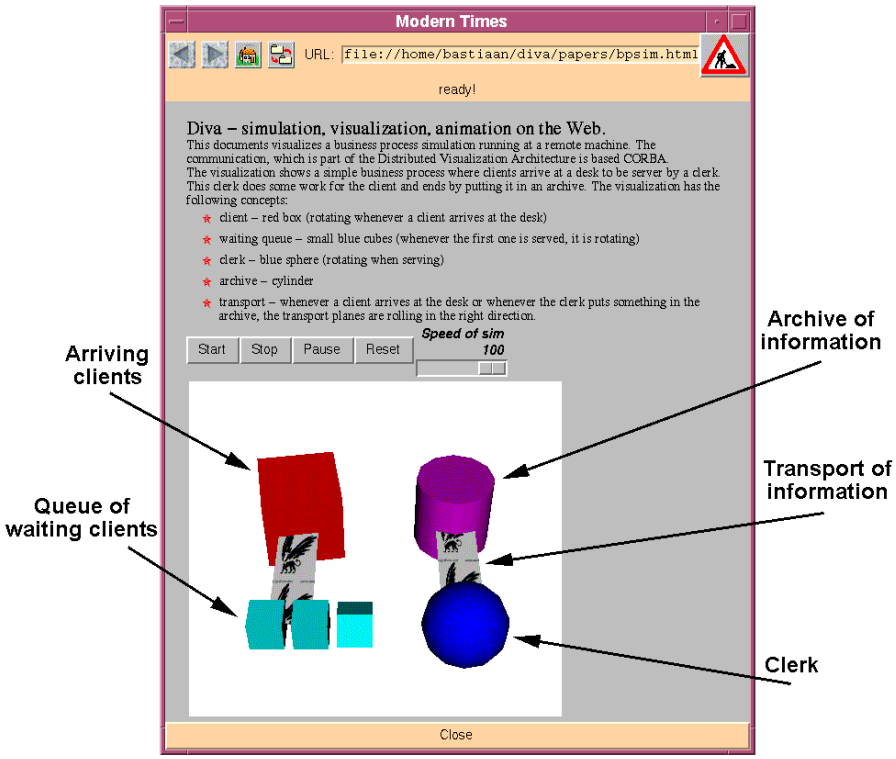


FIGURE 6.2: A screenshot of the Web-browser with embedded business process visualization in the *Modern Times* prototype.

architecture. The Shared Concept Space has been left out and support for multiple users and multiple perspectives is absent. However, the prototype does illustrate how generation and presentation of data can be decoupled and how dynamic information can be distributed in a networked environment.

6.2.2 Technology

Modern Times is based on standard Web-technology and distributed object technology. To distribute text-based information to the Web-browser, it uses a combination of HTML and HTTP. The visualization part deploys CORBA and the Visualization Toolkit to represent the data coming from the simulation server.

(Business process) simulation To generate data for visualization we used the discrete-event simulation library Sim (Bolier & Eliëns 1994). Sim has been written in C++ and thus can easily be wrapped into a CORBA component. An arbitrary simulation model can be programmed in C++ using the Sim library.

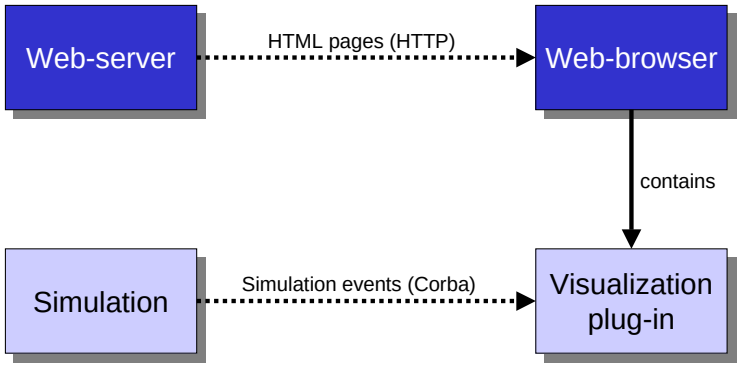


FIGURE 6.3: The software architecture of *Modern Times* comprises four major components.

However, event-graph simulations can easily be specified using a text-based specification.

In addition to Sim, we are using BPSim (Eliëns et al. 1996) which is a business process simulation library built on top of Sim. BPSim is based on the *logistics-based modeling* method (Gerrits 1995), offering high-level primitive entities such as *operation*, *waitqueue* and *employee*.

Visualization The visualization of the simulation is created by a presentation plug-in in the Web-browser (Eliëns et al. 1997). The plug-in maps simulation events to animation in the visualization, such as rotating objects and working conveyer belts. The visualization component uses the Visualization Toolkit (VTK) (Schroeder et al. 1996) to represent the simulation in 3D using OpenGL.

CORBA *Modern Times* is designed as a distributed object oriented system. This implies that it is based on software components which are distributed over a network. The components provide an interface offering services to other objects.

The Object Management Group's (OMG) *Common Object Request Broker Architecture* (CORBA) is an architecture to create applications using distributed objects. The distributed objects can be used like local objects and are accessed through a well defined interface, the *interface definition language* (IDL). The communication between the components of the distributed application takes place via an *object request broker* (ORB).

CORBA abstracts from hardware, operating system and programming language. It enables, for example, the connection of C++ components on Unix machines with Java components on Windows machines.

The *Modern Times* prototype deploys CORBA to connect the simulation to the visualization component. This enabled us to decouple visualization from simulation and to run the software on a multitude of platforms.

6.2.3 Wrap up

Modern Times is the first DIVA prototype. It deploys the Web and distributed object technology (CORBA) to implement a system comprising a Web-server, simulation server, Web-browser and a visualization plug-in. To present the simulation a factory metaphor is used, which closely resembles the original simulation model. And, although *Modern Times* has a simpler software architecture than the architecture finally developed for DIVA, it proves how simulation and visualization can be separated while the visualization still represents the current status of the simulation.

6.3 So Many Users – So Many Perspectives

Collaborative visualization systems are a subset of computer-supported cooperative work (CSCW) applications in which control over the visualization is shared. Elvins & Johnson (1998) categorize collaborative visualization applications by the level of shared control. The following discussion is based on, and contains terminology from, (Elvins & Johnson 1998). As the simplest form they determine **local control** in which a collaborative visualization broadcasts the visual structures (images) to the other participants. Since the users are reduced to passive viewers, external means, such as telephone or teleconference, must be used to exchange feedback.

A more complex variation is **local control with shared data**. Participants can share data from any step in the visualization process while interaction still occurs locally. A third category, called **limited shared control** refers to applications in which participants have a shared view. Cooperative control is limited to annotation of the resulting visual elements as well as control and sharing of the viewpoints. Finally, the most advanced category is characterized by **fully shared control**. Here, any aspect in the process of visualization, from raw data to views, can be controlled as a shared activity.

In the basic software architecture of DIVA, multiple users can have their own presentation model (perspective) while sharing a common resource (local control with shared data). However, in this approach the different users have the feeling that they are the only user of the shared resource. There is not yet support that allows a user to be aware of other users or to interact with them. The goal of *The Great Dictator* experiment is to expand the architecture to support users to collaborate with each other. Here, 'to collaborate' means that the users are able to discuss and interact with visualized information in order to reach a decision (fully shared control).

This section addresses the issues that are involved in this notion of collaborative visualization. In Section 6.4 an extended software architecture is discussed which incorporates most of those issues. Section 6.5 briefly shows an application of collaborative visualization as made possible by an extended DIVA. Finally, Section 6.6 will discuss the technology underlying *the great dictator* prototype.

DIVA focuses on visualizing information from different perspectives. We can distinguish between two distinct phases of activities within this approach. The first phase is to define and experiment with the perspectives. This activity is done mostly in solitude, although multiple users can share a primary model or a derived model. The purpose is to determine the information need and the relevant data for that need. The second phase is that of multiple users collaborating by reviewing and discussing the different defined perspectives. *The Great Dictator* focuses on the latter, the collaboration phase.

When a group of people collaborates, the group members must share a common workspace (Ellis et al. 1991). The task of a group of users is to interact with each other and present different views on shared information. Let us assume that the goal is to reach a decision, for instance, concerning which strategy to follow in the upcoming year to improve a particular business process.

6.3.1 Sessions

Collaboration normally takes place in some kind of meeting, which can differ in interaction protocol, group size, formality, etcetera. Each participant of the collaboration can have one or more roles depending on the sort of meeting. A **role** is a set of rights and obligations (Ellis et al. 1991). We distinguish the following roles: chair, listener, talker and interactor. The **chair** sets up the session, a **listener** is a passive participant, a **talker** is actively explaining his arguments and, finally, an **interactor** is able to interact with shared resources. The rights and obligations of the different roles are determined by the **interaction protocol**. The possibility to switch roles dynamically is important, since a listener can change into a talker from one moment to the other.

Collaborative visualization in DIVA is a virtual meeting, where the participants are at different places and their desktops are connected by a network. We will call the event of such a virtual meeting a **session**. Session management should support several kinds of sessions and thus be able to handle changing numbers of participants, their roles and interaction protocols.

The notion of **subgroups** makes it feasible to split the total group of participants in (non disjoint) groups. These subgroups can communicate separately or perform subtasks. Subgroups can come into existence dynamically.

6.3.2 Sharing perspectives

It is important that the cooperators can show their personal perspective or view to other participants, in order to support their arguments in a discussion. One way to share views is for one participant to **enforce** his perspective onto another user or group. Views can also be shared by means of a **perspective repository**, where participants can select a perspective they would like to consider. The perspectives they can choose from, must be deposited by other participants. This implies that not every participant should have to create her own perspective before joining a session. Obviously, there is a need to maintain meta-information, explaining what the perspectives are about.

6.3.3 Interference versus non-interference

The common basis for the collaborators is the primary model, for instance, embodied by a simulation. Several derived models can depend on the primary model, and each derived model could be related to a number of presentation models. When collaborating, the common basis should be the same for all the cooperators at every moment in time. To assure consistency, it is best to have the simulation act autonomously without the slightest interference. We will refer to this as **non-interference**. Non-interference does not restrict the possibility for each user to create his own perspective in any way. It does prevent somebody from rewinding, changing parameters or restarting the simulation while others do not want or expect this.

However, the need to stop, rewind or change parameters in a simulation is obvious. Considering multiple *what-if* situations, for example, is necessary when looking at different re-design alternatives, each with its own set of parameters. One way to meet this requirement, while upholding non-interference, is to store all past events in a database or to allow copies with different parameters of the simulation to be started.

While interaction with the primary model should be avoided as much as possible, derived models can be used in a more flexible manner. Several derived models can be created, all depending on one primary model. While the primary model can be considered a common basis that should not be interfered with, the derived model can be seen as a common workspace that permits **interference**. All participants of a session could use the same derived model or multiple derived models could be created, depending on the need to share information concepts or to be independent of the other users.

6.3.4 Communications

Some form of user-to-user communication is necessary to enable collaboration. These communications can range from a simple chat tool or whiteboard to sophisticated audio/video conferencing tools. Tools of interest for use with DIVA

include telepointers, to point out things of interest, raising hands, to indicate someone wants to speak, and voting tools, to support decision making (Ellis et al. 1991). *The Great Dictator*, however, focuses on the visualization part of collaboration (sharing perspectives) and leaves direct user-to-user communication open to other or further research activities.

6.3.5 Requirements

Taking into account the issues for collaborative visualization mentioned above, we can summarize the following requirements.

1. **Session management is needed to control the virtual meetings, including the participants, their roles and interaction protocols.**
2. **The participants must be able to share their perspectives by enforcement and via perspective repositories.**
3. **The primary model should be interfered with as little as possible.**
4. **Additional communication support is necessary, but falls outside the scope of this experiment.**

6.4 Collaborative Visualization Architecture

The DIVA architecture is intended as an open software architecture and should be flexible enough to incorporate new components. In *The Great Dictator* experiment, we had to extend the basic DIVA software architecture with collaboration capabilities. How we adapted the basic software architecture to allow for collaborative visualization sessions and sharable perspectives is the topic of this section. Related collaborative architectures are described in Bentley, Rodden, Sawyer & Sommerville (1994) and Reinhard, Schweizer & Völksen (1994).

6.4.1 Software components

Figure 6.4 shows the main components of the software architecture of *The Great Dictator*. In the following list, the three components that were already present in the basic DIVA architecture are listed first. The last four components in the list extend DIVA. After the list, a short description is given for each of the new components.

- **Data Provider** — the simulation providing data.

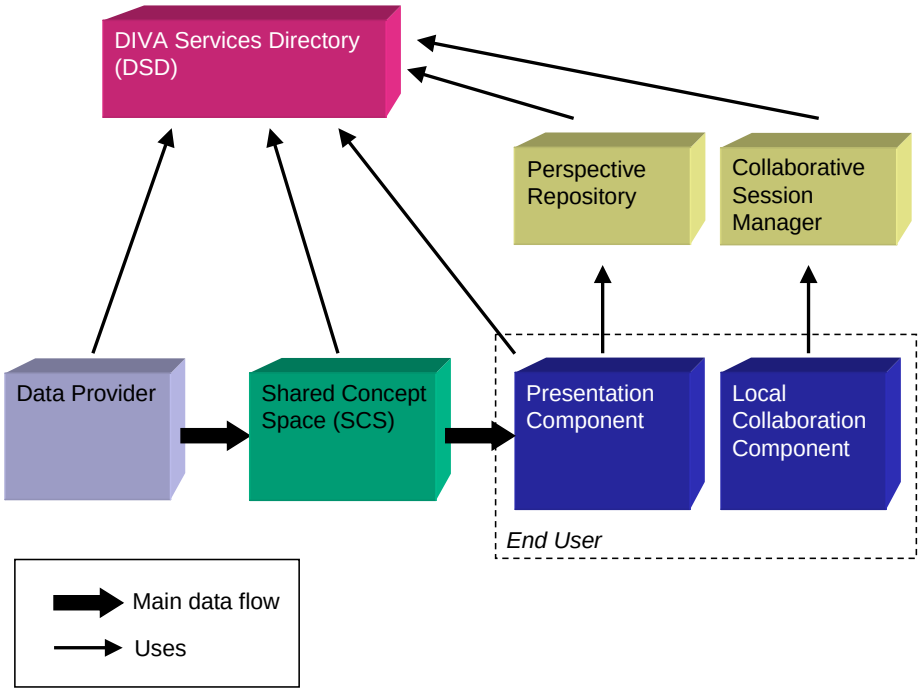


FIGURE 6.4: The main components of the collaboration prototype.

- **Shared Concept Space** — communication infrastructure to exchange the information.
- **Presentation Component** — the actual information visualization.
- **Diva Services Directory** — central registering of components.
- **Collaborative Session Manager** — overall coordination of virtual meetings.
- **Local Collaboration Component** — local collaboration support.
- **Perspective Repository** — shared repository for visualization perspectives.

The **Diva Services Directory** (DSD) is the central directory component. DIVA components (or services) can register here, identifying themselves and giving their location. Once they are registered, the DSD can inform other objects about the availability and whereabouts of these services.

The **Collaboration Session Manager** is responsible for the overall coordination of the virtual meetings (sessions). It deals with interaction protocols, which

means it knows about the participants and their roles, sharing perspectives, user to user communication and consistency.

The **Local Collaboration Component** is directly connected to the session manager. It is present at each participant's desktop and handles interactions and information related to a collaborative session. It may, for example, display a list of participants and offer communication facilities.

The **Perspective Repository** is a container for visualization perspectives. Users can deposit their means to visualize data in the perspective repository. This allows them to reuse visualization perspectives or to make them available to other users.

6.4.2 User environment

Figure 6.5 shows a typical user environment. Outside of the user environment, two components are shown. A data provider, which is a simulation in this figure, feeds data into the Shared Concept Space. This is depicted by the fat arrow. These two components can be situated anywhere on the network.

The user environment, which is located at the user's local machine, consists of non-visual components (the sim control, display agents, local collaboration component) and components that do use the display of the user. For example, the local collaboration component makes use of two separate user-interface components. One component is intended to display information about the collaboration sessions and the other is used for user to user communication.

Controllers

Every DIVA component can have a separate mobile **controller**. It can be moved from one user environment to another, so it can be shared by several participants. The ability to use a controller depends on the role of the user. Participants can request a controller or the chair could appoint it to one of them.

Controllers can have several functions. For example, a simulation can be controlled by starting and stopping the simulation and changing certain parameters. A controller for a Shared Concept Space can be used to create new derived concepts or to decide which data from the generator is selected.

Display agents and gadgets

From the Shared Concept Space, information is being sent to **display agents**. These agents are present at the user environment and each of them maintains one gadget. A **gadget** is a software component capable of displaying a visualization primitive. As an example, consider the visualization in Figure 6.6 on

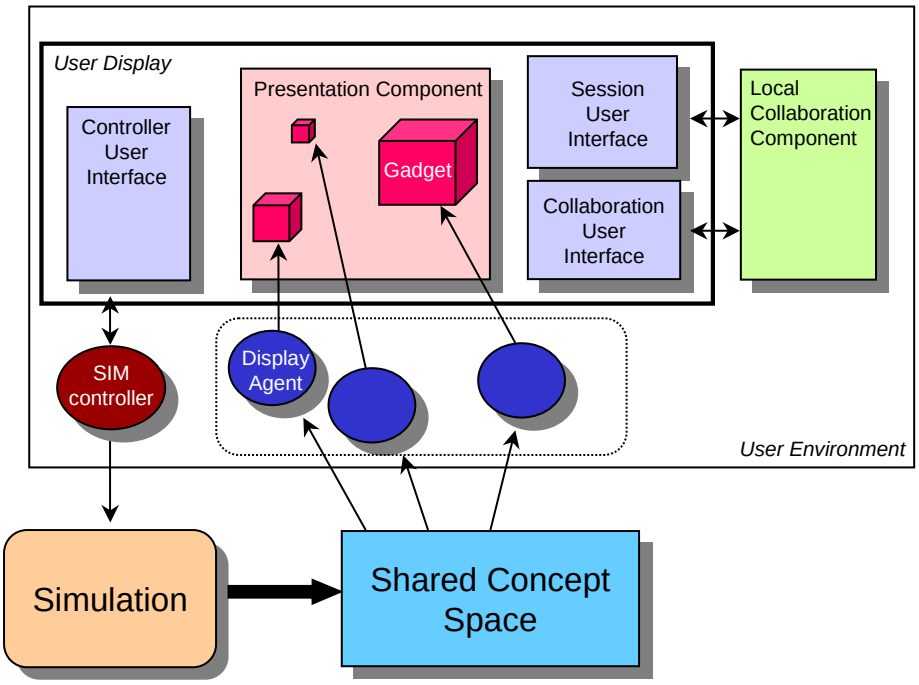


FIGURE 6.5: The user environment shows how information in the Shared Concept Space is visualized using display agents and gadgets. Additionally, it shows how users can interact with the simulation (control interface) and cooperate with the other participants in the collaboration session.

page 107. The line of puppets in front of the desk is a visualization gadget depicting a queue. When the display agent senses that the length of the queue increases by one, it accordingly places an additional puppet on the screen.

In addition to being located within the Presentation Component, display agents also reside in the Perspective Repository. When a user requests a certain view from the repository, the agents that represent that view are cloned and moved to the user environment to show the perspective in the visualization. Enforcing a perspective onto another user is accomplished by cloning and moving the display agents directly from one user's environment to the other's.

6.5 Application of Collaborative Visualization

Figure 6.6 contains a screenshot of the desktop of a decision maker participating in a collaborative session. We describe a scenario of how the user gets to this display.

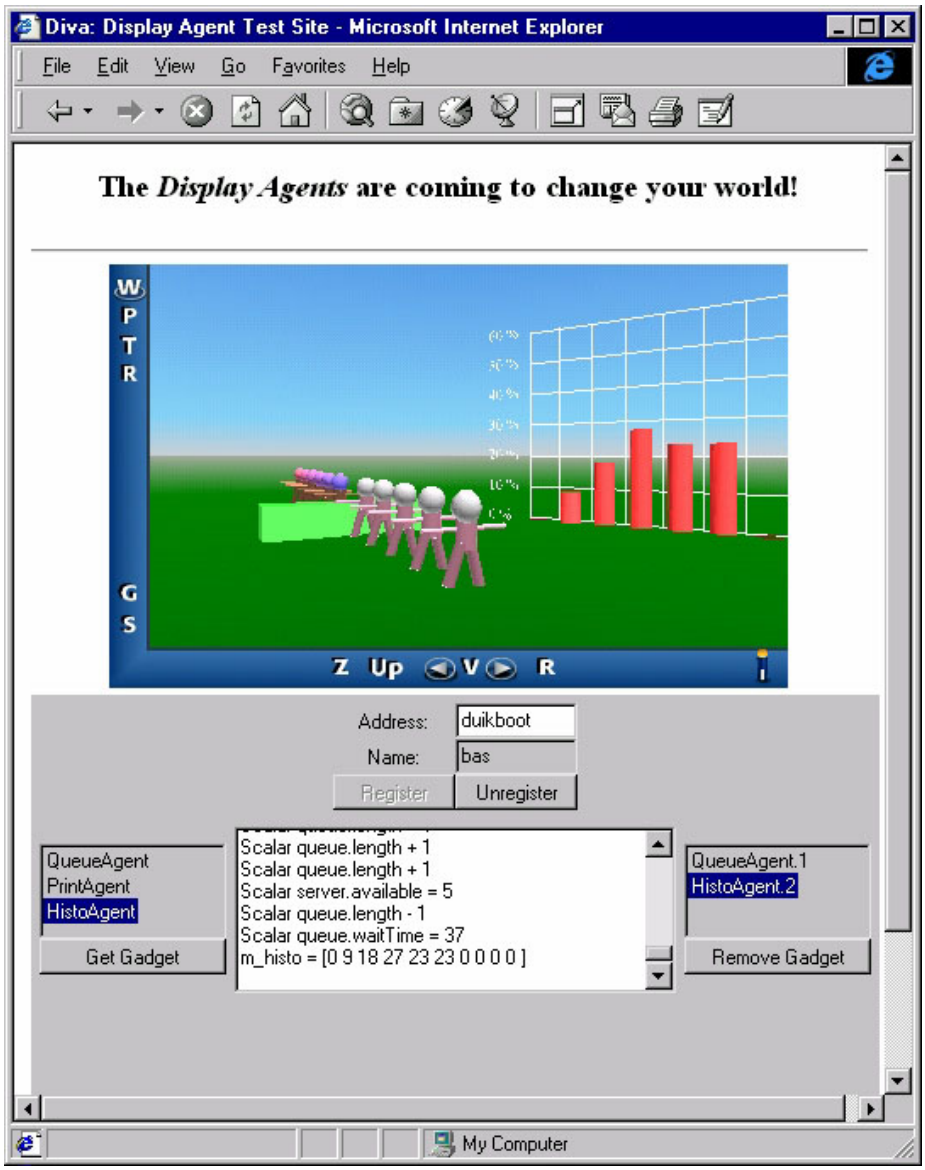


FIGURE 6.6: Screenshot of The Great Dictator prototype.

The decision maker starts a Java and VRML enabled Web-browser and follows a link pointing to a DIVA server. The resulting HTML file will setup the user environment. First, the user has to log in, making available her name and network address, and after that she can choose from one or more sessions to join. Once the user enters a session, she will be assigned a role and then gets

a default or enforced perspective. The VRML world showing her view is embedded in the Web page.

Figure 6.6 shows how the VRML browser Intervista's Worldview is integrated into the Internet Explorer Web browser. The VRML world consists of two visualization primitives (gadgets): a queue depicting the number of clients and clerks currently involved. The other gadget is a histogram agent, which gives an indication of how long people have to wait before being served.

The bottom of the browser contains the Java applet that connects with the Diva servers, and takes care of hosting the deployed display agents. When information in the Shared Concept Space is modified, e.g. because a client arrives in the simulation, or because the number of clerks is changed manually, the display agents will get updated about the change and will accordingly update the VRML world.

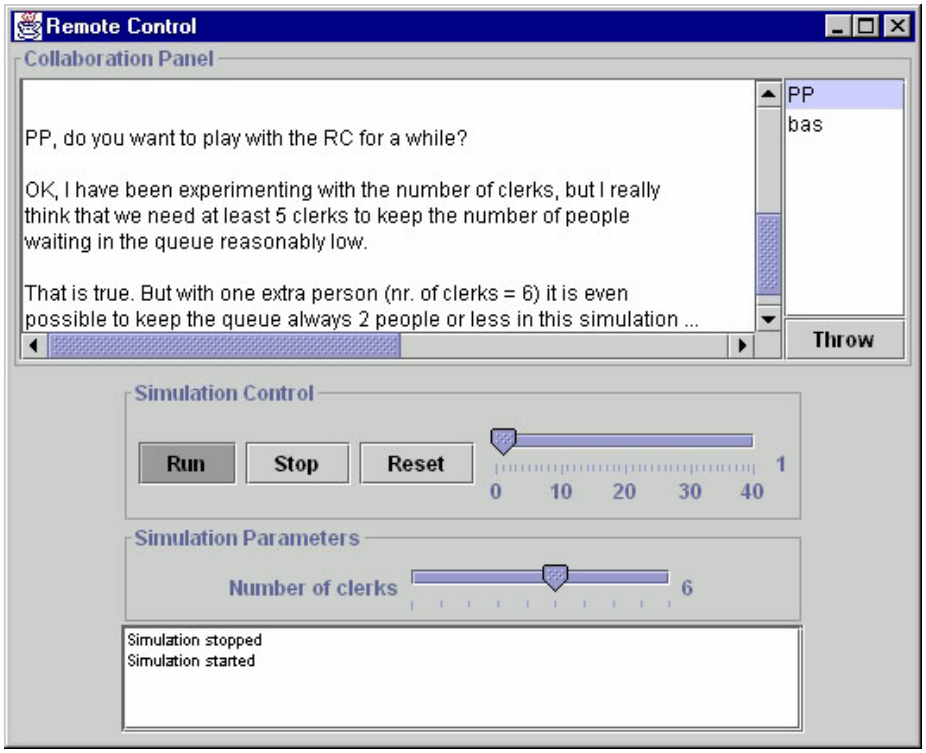


FIGURE 6.7: Users can control the simulation by means of a single shared remote control

To interact with the business process simulation, the decision maker can request a remote control, which will arrive at her environment (assuming that she is allowed to do so). Once the remote control arrives in the form of a mobile object, this object pops up a remote control user interface on the display.

The user is then able to control the simulation that is associated with the remote control. Using the remote control shown in Figure 6.7, people are able to control the simulation (start, stop, reset) and modify the parameters of the simulation (in this case the number of clerks behind the desk). Additionally, users can type in messages in a collaboration panel. Other users of the remote control can then see, what previous users have done. The remote control can be transferred to other users by selecting a person from the participant list on the right and pressing the 'Throw' button.

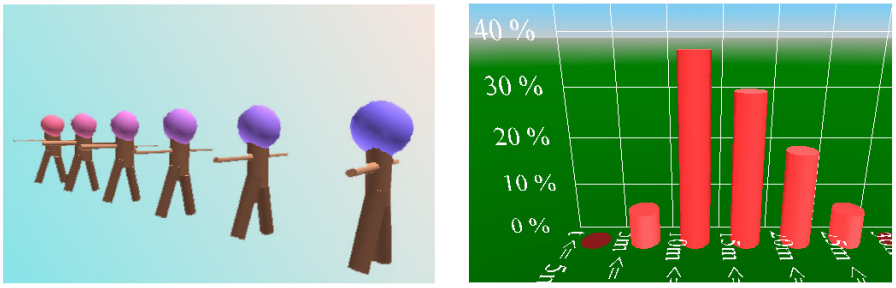


FIGURE 6.8: Two perspectives are used in the TGD prototype. Left: the queue gadget shows the number of people currently waiting in the queue. Right: a histogram gives an overview of how long people had to wait in the queue.

Figure 6.8 shows the two deployed perspectives of TGD. The waiting queue represents the number of people currently waiting to be served in the simulation. The right-hand side of Figure 6.8 contains an alternative visualization of the same simulation. This perspective shows how long people had to wait over time instead of depicting the length of the current queue. The first column of Figure 6.8 depicts the percentage of people waiting shorter than 5 minutes, the second one the percentage waiting between 5 and 10 minutes, then 10 to 15 minutes, etcetera.

6.6 TGD Technology: dynamic and mobile VRML gadgets

To share and present visualizations in *The Great Dictator* prototype, a combination of distributed object technology and the Virtual Reality Modeling Language (VRML) (ISO 1997) has been deployed. VRML is a language to describe 3D virtual worlds for use on the Web. However, VRML still falls short when it comes to support dynamic, interactive multi-user worlds and shared visualizations. In this section, we will therefore discuss the use of CORBA and mobile object technology for the realization of VRML gadgets and VRML dis-

play agents that allow for updates in response to server-push events as it has been used in the TGD experiment.

6.6.1 Background on VRML

In June, 1994, VRML was born. It started as a specification to open the road towards platform independent three-dimensional graphics on the Internet. However, in one of the first VRML books around, Mark Pesce states that VRML was by no means finished (Pesce 1995). New features would be included in future specifications.

The most intriguing features promised were support for **dynamic** and **interactive** worlds. Dynamic worlds should be updated whenever new information is available by means of server-side push. Interactivity is necessary to create a *real cyberspace*. According to Pesce, this will probably be achieved using a separate controlling application which updates the VRML world using an *application programming interface* (API).

As we know by now, two years later VRML 2.0 – the moving worlds came into existence (later this became the ISO VRML97 standard (ISO 1997)). The most appealing addition to VRML 2.0 is the support for motion and interactivity in the up till then static realm of VRML. Objects can be modified, rotated, and moved by means of `Interpolator` nodes. In addition to this, VRML nodes respond to user-interaction such as clicking and dragging by means of `Sensor` nodes. More complex actions and interactivity are covered by `Script` nodes which inline pieces of code, e.g. Java classes or JavaScript, into a VRML world. The connection mechanism between all mentioned nodes are `ROUTE`s which guide events from node to node passing triggers and information through the scene.

So, is the specification complete now? Or do we still require additional features? Well, we still don't have dynamic worlds in the sense that a world is updated whenever data on the server changes. Additionally, multi-user support is still lacking: you are the only inhabitant of a VRML world; unless you use proprietary extensions such as blaxxun (Blaxxun Interactive 1997).

And what about the promised API to control the VRML world externally? The **external authoring interface (eai)** —which is not part of the standard, but nevertheless almost always implemented by VRML browsers— is exactly this. Java applets can control downloaded VRML scenes by means of a Java API on the VRML browser.

Luckily, the external authoring interface is powerful enough to allow us to implement the lacking features of VRML 2.0. How? That is the topic of the remainder of this section.

6.6.2 Dynamic updates

A VRML gadget is a combination of a VRML prototype, possibly including Script nodes, and a Java object controlling the prototype by means of the external authoring interface. Gadgets are separated but coherent pieces of a visualization. For example, the perspectives in Figure 6.8 are implemented using a `QueueGadget` and a `HistogramGadget`.

Because VRML gadgets contain external Java code controlling the VRML world, gadgets can easily be extended to listen to events broadcast by servers. For example, the Common Object Request Broker Architecture (CORBA) (Siegel 1996) allows remote calls between objects written in different programming languages. Special services, such as the event service, allow a server to broadcast events to multiple receivers. In the DIVA architecture, we can rely on the Shared Concept Space as our model of communication between information provider and visualizer.

All gadgets, possibly running on multiple machines, update the VRML scene according to the received events. Whenever, when the simulation of our example broadcasts that a new client has arrived, all QueueGadgets increase the number of puppets in the VRML scene by one.

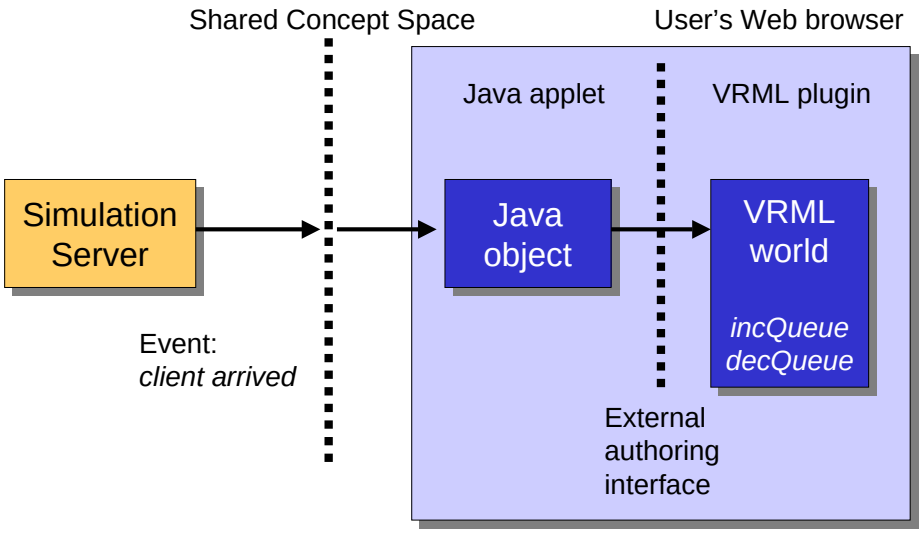


FIGURE 6.9: A *client arrived* event is sent to Java objects which update the VRML worlds

Figure 6.9 illustrates how dynamic updates work in our system. On the left is the simulation server which produces the information published via the Shared Concept Space. On the right, is the user's Web browser comprising the Java controlling applet and the displayed VRML world. After the simulation server has published the event, a Java object running in the Web browser receives

the event. It responds by activating the `incQueue` event on the VRML code which adds a puppet in the VRML world. A VRML snippet sketching the QueueGadget prototype is given below:

```
#VRML V2.0 utf8
PROTO QueueGadget [
  eventIn SFInt32 incQueue
  eventIn SFInt32 decQueue
  eventIn SFInt32 assignQueue
] {
  Group {
    # additional VRML scenery goes here
    # the waiting people
    DEF CLIENT Group {
      children [
        # filled by script defined below
      ]
    }
    DEF DYNAMICS Script {
      directOutput TRUE
      eventIn SFInt32 incQueue IS incQueue
      eventIn SFInt32 decQueue IS decQueue
      eventIn SFInt32
            assignQueue IS assignQueue
      field SFNode clients USE CLIENTS
      url "Dynamics.class"
    }
  }  # end group
}
```

As we can see from the code above, the number of people displayed in the waiting queue is controlled by sending events to the instance of `PROTO Queue-gadget`. An `incQueue` event increases the queue by the supplied number, a `decQueue` decreases whereas `assignQueue` gives the queue a specified length. The incoming events are handled by a piece of Java code which is inlined by means of the `Script` node. The Java script node takes care of adding the puppets as children to the `CLIENT` group.

To be able to identify to which information fragment an update-message belongs, the information from the simulation has been structured hierarchically. For example, in the waiting queue example, the *queue* concept contains all events concerning the current length of the queue, whereas the *clerk* concept contains information about the number and performance of clerks serving behind the desk. A schematic overview of a part of the used Shared Concept Space is given below:

- *queue*
  - *length*: length of the waiting queue
  - *waitTime*: waiting time of last client
- *clerk*
  - *available*: total number of clerks available
  - *working*: number of clerks serving

Whenever information is updated at the server side, an update-event must be distributed via the Shared Concept Space. For example, when a client arrives in the simulation, increasing the current queue length by one, the server publishes "`queue.length increases by 1`."

The listeners (VRML gadgets) listen to one or more subchannels of the hierarchical concept space. For example, the discussed queue gadget listens to the *queue* concepts only. Whenever it sees a new update-event concerning the length of the queue it invokes an event on the VRML world via the external authoring interface. For example, on receiving the "`queue.length increases by 1`" event it invokes `incQueue(1)` on the `queueGadget`. Consequently, the Java `Script` node (which is part of the VRML prototype) adds a new puppet to the visible queue of waiting people. Thus, updated information at the server is propagated by means of events to reflect the change in the VRML visualization.

6.6.3 Mobile VRML gadgets

An emerging trend in distributed object oriented programming is the notion of **mobile objects** (see for example (ObjectSpace 1998) and (Baumann, Hohl, Rothermel, Schwehm & Strasser 1998)). A mobile object can migrate from machine to machine taking functionality, data and status as it moves. Mobile objects are especially useful in providing adequate support for developing and deploying collaborative applications. For example, the exchange of VRML gadgets allows users to discuss different visualization perspectives. Additionally, mobile objects enable the creation of repositories of VRML gadgets. We call VRML gadgets that are based on mobile object technology **VRML display agents** .

Whenever users want to employ display agents to visualize the simulation, they connect to a repository (display agent server) and request a clone of a display agent. As soon as the display agent arrives, it docks in the Java virtual machine of the Web browser. Additionally, it downloads the PROTO gadget and inserts an instance of the gadget into the VRML world. After that, it modifies the VRML object to match the current status of the visualization (a mobile object maintains status while migrating). Finally, it starts listening again to events coming from the server to keep up to date.

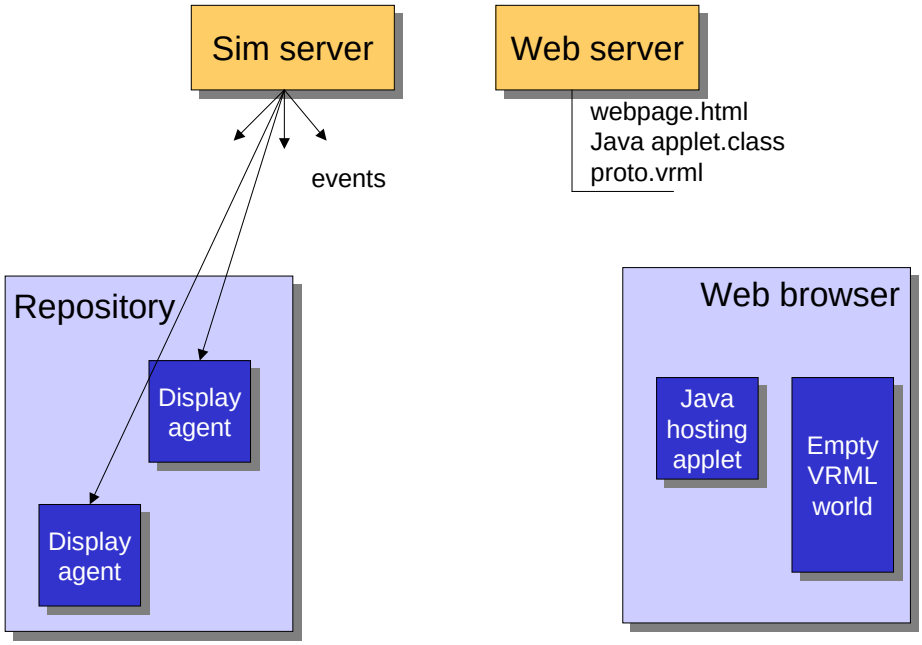


FIGURE 6.10: The user has downloaded the applet but still has an empty VRML world

Figures 6.10 and 6.11 illustrate the architecture comprising the display agent repository, the simulation server, the Web server and a client machine with a Web and VRML browser. When a user decides to start a visualization he or she downloads an HTML page containing the VRML plugin and the Java hosting applet. At that stage, the VRML world is still completely empty (Figure 6.10).

After all necessary components have been downloaded, the hosting applet connects to the agent repository and shows a list of available gadgets (step 1 in Figure 6.11). By selecting a gadget and pressing the `Get Gadget` button a clone of the selected agent is created and moved to the client machine (step 2). Here, the display agent uses the external authoring interface to download the gadget PROTO by means of adding an EXTERNPROTO (step 3). As a result, an instance of the gadget has been created in the VRML browser. Update-events broadcast by the simulation server are now received by both the original agent in the repository and the cloned agent in the user's browser. On each broadcast event, the Java agent calls the appropriate event on the VRML prototype instance to reflect the change in the data at the server (step 4).

This approach is flexible. New VRML gadgets can be added dynamically to the repository and Web-server and users can immediately employ them without reloading or restarting existing applets or VRML worlds.

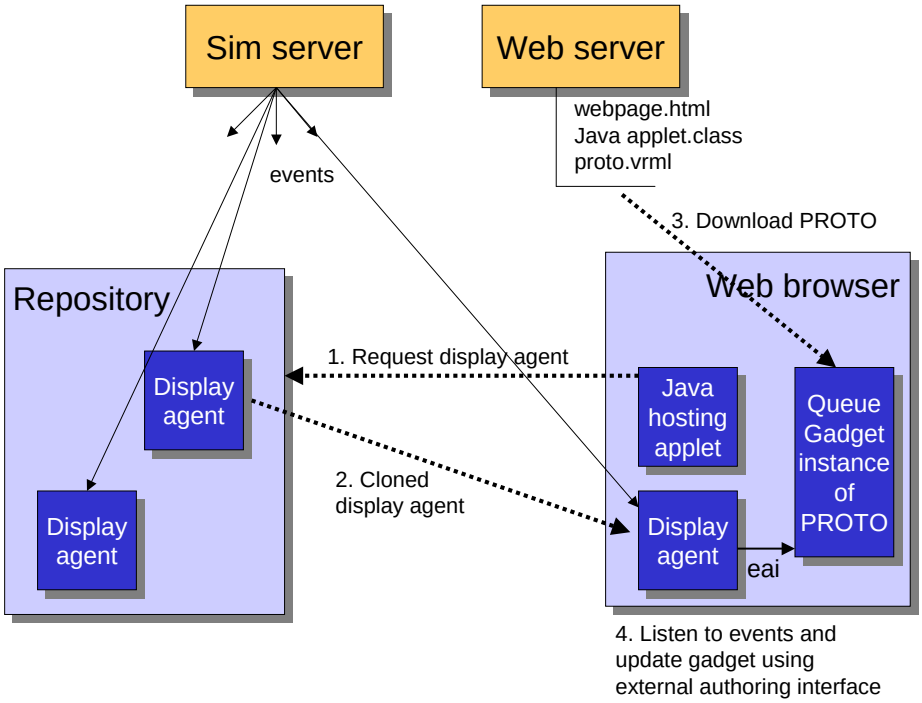


FIGURE 6.11: The applet connects to the repository and downloads a display agent containing the knowledge to visualize the events

6.6.4 Implementation of display agents

Voyager (ObjectSpace 1998) is an agent ORB written in Java, which supports CORBA. Voyager allows us to use mobile objects, a feature which CORBA does not have. For example, we use Voyager's agent construct to realize the mobile controller components. These components are able to dock at a user environment and can subsequently show their user interface on the screen to let the user interact with remote information providers such as simulation servers.

Additionally, we deploy Voyager to implement the display agents. The **DisplayAgent** class extends the **Agent** class provided by Voyager. This automatically makes display agents clonable and movable. An outline of the interface of the display agent class is given below:

```
interface DisplayAgent extends Agent
{
  public void launch(DAServer server);
  public void gotoApplet(DAApplet applet);
  public void setSpace(String spaceName);
};
```

The `launch()` method is called once, just after the display agent is created. It moves the display agent to the indicated agent server (repository), where it will reside for the time it is alive. The `gotoApplet()` method is called by the server when the display agent server receives a request from an applet to send a clone of that display agent to a user's machine. The `setSpace()` method is invoked by the launcher or the agent to tell the agent to which space it has to listen to receive the appropriate update-events.

Our experiences with Voyager and the mobile object technology are mixed. On the one hand, the concept of moving objects (including both code and data) around a system is great. It allows for a direct reflection of the design onto the implementation. On the other hand, debugging is horrible. Sometimes mobile objects get lost and it is very difficult to see whether that is the fault of the server, the client or the network in between. Summarizing, mobile object technology has great potential but probably needs to become somewhat more mature before it can be used in critical systems.

6.7 Summary and Conclusions

An important element of DIVA is its software architecture. This software architecture was not designed in a single attempt, but merely evolved through a series of experiments and case studies. This chapter provided an overview of all experiments and, subsequently, described two prototypes in more detail: *Modern Times* and *The Great Dictator*.

Modern Times deploys the Web and distributed object technology to implement a system comprising a Web-server, simulation server, Web-browser and a visualization plug-in. By means of *Modern Times* we have shown how information generation and presentation can be separated in a distributed environment.

In *The Great Dictator*, we have shown an implementation of most of the basic DIVA software architecture. However, *The Great Dictator* extends DIVA with collaboration features. By means of collaborative visualization, decision makers are able to discuss a shared information source, such as a business process simulation, to convince other people of their point of view. To achieve this in *The Great Dictator*, we require the possibility of sharing perspectives through enforcement and repositories.

The software architecture of *The Great Dictator* achieves the exchange of perspectives by means of cloning and transporting display agents, which in turn define how and what information is presented to the user. To manage the cooperative sessions, we have two collaboration components that handle the rights and obligations belonging to the roles of the participants.

In the prototype implementation of *The Great Dictator*, we have deployed VRML as the basis tool for visualization. To be able to implement all the

features needed for collaborative visualization, we have combined mobile object technology and VRML prototypes into VRML gadgets and VRML display agents.

A design requirement for the DIVA software architecture was that it must be open to extensions such as interaction and collaboration. In *The Great Dictator* case study we have shown that this is indeed possible. The basic architecture (information provider, Shared Concept Space and information visualizer) is still present in the software architecture of TGD. Extensions are structured in the form of additional software components. Collaboration components enable participants to cooperate in a visualization session. The perspective repository allows for a central store of display agents that enables the exchange of visualization perspectives amongst participants.

CHAPTER 7

3D Gadgets for Business Process Visualization

Wait till the Quake^{tm} generation has reached the management level.
Anton Eliëns.

Chapter 3 introduced BizViz (Business Visualization) as a means to monitor and control business processes at Gak Netherlands. The visualizations designed for that project are uncomplicated and easily accessible. As a final experiment in the range of case studies as described in Chapter 6, this chapter discusses a 3D-remake of the Gak business process visualizations. Based on our experiences with both 2D and 3D visualizations within the same domain, the chapter also provides a discussion of 2D versus 3D visualizations to support decision makers in a business context.

7.1 3D BizViz

BizViz is becoming increasingly important, since managers recognize the power of human visual intuition in information-rich decision tasks. Nevertheless, despite its promises, 3D visualizations are far less common than one would expect.

This chapter describes an exploration where we took the 2D visualizations of Chapter 3 as a starting point, for which we subsequently provided a 3D visualization. We introduce a small collection of 3D visualization gadgets and associated behaviors, implemented in Java3D, which proved to be relatively complete for this case.

For each of these gadgets and behaviors, we discuss requirements and design trade-offs. The experiment illustrates the usability of our gadgets and their associated behaviors in an actual business process in the domain of social security.

Structure Section 7.2 refreshes the reader's memory of the 2D visualization experiment as described in Chapter 3. Section 7.3 presents the reusable collection of Java3D gadgets we deployed to create the 3D prototype. The collection consists of both behavior and visualization components. The application to visualize business information is described using the case study at Asz/Gak in Section 7.4. Finally, in Section 7.5 we end with conclusions.

7.2 Managing Business Processes at Gak NL

Gak is the largest social security provider of the Netherlands. One of the core businesses of Gak NL is processing applications for benefits. For example, when people are unable to work due to illness, they can go to the Gak and apply for a benefit. Their applications are processed in a number of stages, including medical inspection and ability assessment. This process is the business process that we will use as the topic of our information visualization later.

Chapter 3 discussed a 2D visualization system to support managers who control the benefit application processes. The system was built in two phases. First, we concentrated on increasing insight by creating visualizations to indicate the bottlenecks in the business process. The data sources consisted of databases containing measurements of the time that applications needed at the stages of the process. In the second phase of the project, we created a business process simulation and used the previously designed visualizations to display the results of that simulation.

Figure 7.1 contains a screenshot of one of the visualizations created to increase insight in the current production status. The combination of process structure and associated histograms helps managers to quickly find bottlenecks. The histograms indicate the status of applications, that is whether they are early, in time or too late. In addition to this, visualizations are available to give an overview of the past in order to search for trends. As a third option, managers are able to asses the results of possible interventions such as adding people to the workforce by means of simulation. In summary, the prototype uses the same visualizations to view past, present and future but requires multiple windows to present them.

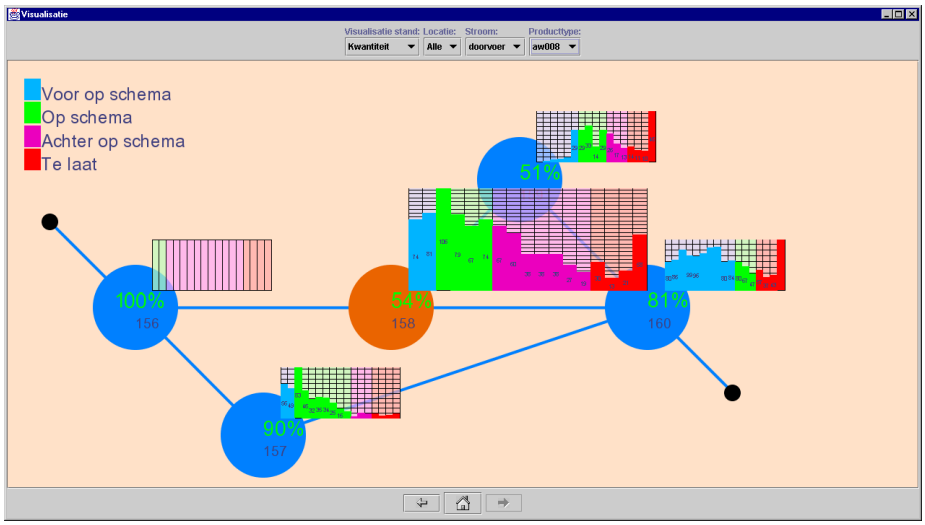


FIGURE 7.1: 2D visualization of the throughput of the business process. (reprint of Figure 3.2)

Based on our experiences with the two-dimensional visualizations, the experiment described in this chapter produces 3D version of business visualization. Since the future users have little experience with 3D, the handling and navigation of the three-dimensional information space will almost certainly be less effective than its 2D counterpart. However, 3D has the potential of high information density. This gives us the opportunity to catch more information, e.g. concerning past, present and future, in a single environment.

7.3 Visualization Gadgets in Java3D

Java3D (Sowizral, Rushforth & Deering 1997) is the 3D application program interface (API) of the Java language. It is used to create platform-independent 3D applications that can be used over the internet. Additionally, Java3D can read and display VRML files and combine VRML scenes with Java3D contents.

Although Java3D offers some high-level building blocks, such as built-in primitives (sphere, cone, etcetera) and behaviors for interaction, it still requires a lot of programming effort to create a simple visualization. To fill in this gap, we have created a collection of reusable visualization gadgets implemented in Java3D.

The set consists of two types of primitives. First, we have the behaviors which usually do not present themselves graphically, but merely exist to add interaction to a scene graph. The second class of components we have created, the

gadgets or visualization primitives, represents information by means of 3D visualization. The gadgets we describe here are the cone tree, the histogram and the graph.

7.3.1 Overview of behaviors

The DIVA 3D gadgets collection currently contains five different types of behaviors. Two of them (brushing and modify behavior) are discussed in somewhat more detail below. The current behaviors are:

- **BrushingBehavior** reveals extra information about the object that the input device is currently pointing at.
- **KeyBehavior** is a generic behavior class to move objects through the scene according to key presses. It is often used to move the camera viewpoint.
- **MenuBehavior** displays a context-sensitive 3D menu when the user selects an object.
- **ModifyBehavior** assigns multiple manipulation facilities to a single mouse button. Supported examples include rotation, translation, scaling and iconification of groups.
- **TranslateBehavior** translates 3D objects in such a way that they move along the screen's x-axis and y-axis (instead of the default behavior of translation along the object's local coordinate system).

BrushingBehavior

The retrieval of more detailed information about parts of a visualization without changing the visualization itself is called brushing. When a pointing device is moved over a particular component in the visualization, extra information appears on top of the selected object. The advantage of brushing is that information can be retrieved quickly without replacing the current visualization. A simple mouse movement is enough to reveal, for example, the numbers on which the visualization is based.

The goal of the brushing behavior is to allow a program to easily add information in the form of brushing to an existing scene graph. For the brushing behavior we have three requirements:

- It must be possible to add brushing information to an existing scene graph.
- It must be possible to dynamically change the information that is associated with brushable objects.

- The brushing behavior must be relatively efficient, because it is expected to be used often and should therefore not pose too much of a burden on the system.

The brushing behavior we developed satisfies the defined requirements. One can add it to an existing scene graph and dynamically change which 3D objects are brushable and what information is to be associated with them. Figure 7.2 illustrates the effect of brushing a VRML scene that has been extended with brushing capabilities.

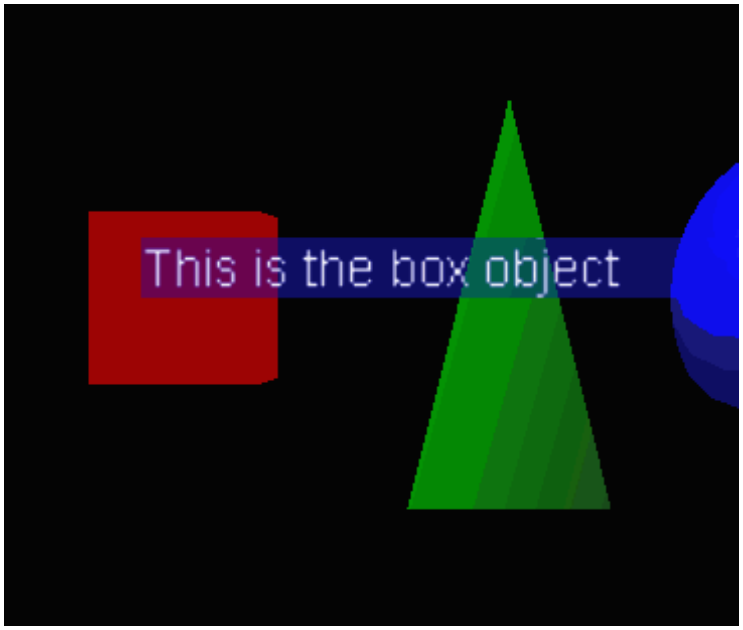


FIGURE 7.2: Brushing the red box.

ModifyBehavior

An important way of interacting with a visualization is the ability to manipulate the objects present in the 3D scene. The standard Java3D behaviors support this by means of a combination of mouse and key presses. A problem with these standard behaviors is that the controls are not intuitive and use all three mouse buttons (many users only have one or two mouse buttons). As an example of a counter-intuitive standard manipulation behavior, the translation behavior moves objects according to their local coordinate system. Depending on the viewpoint and the object's orientation this can be a completely different direction than the user moved the mouse.

The requirements for the modify behavior are:

- The behavior must be selectively applicable. In other words, the program must be able to indicate what objects the user can and cannot manipulate.
- The behavior must be extensible; it should be possible to add additional manipulation methods.
- The object or group of objects under manipulation should be clearly marked.
- The basic rotation, translation and scaling manipulations should work intuitively. For instance, moving an object to the left must result in the object moving to the left along the x-axis of the screen, regardless of the orientation of the object and the current viewpoint.

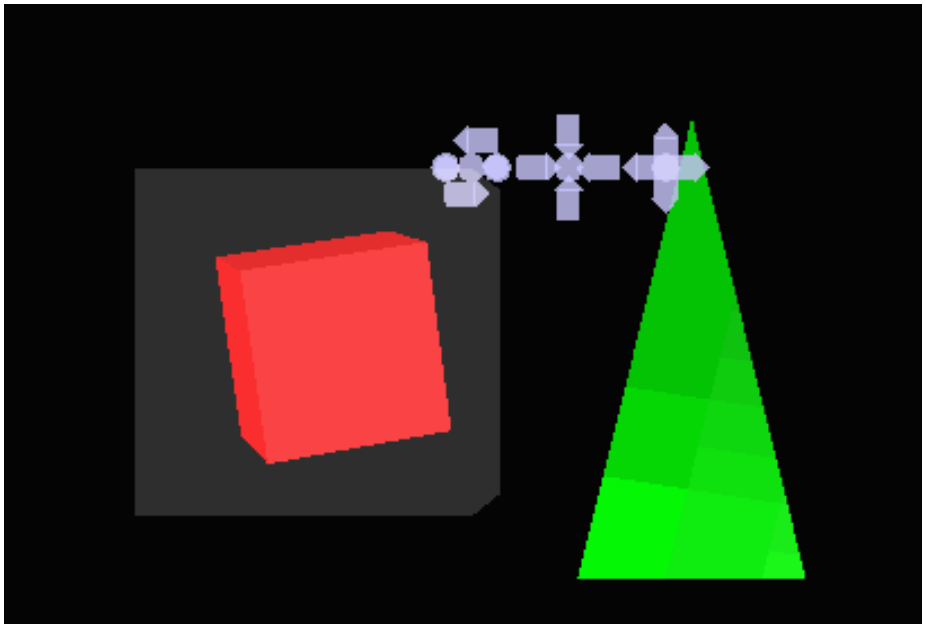


FIGURE 7.3: The red box' modify behavior has been activated.

The modify behavior allows users to pick an object. A transparent box to indicate selection surrounds the selected object. Users can manipulate the selection by pressing one of the icons and dragging the mouse.

Figure 7.3 shows a scene containing a box and a pyramid. The box object has been selected and rotated a little. The selection of the red box is visible through the transparant white box drawn around it. The three little icons indicate three possible actions. From left to right the icons can be used for respectively rotation, scaling and translation. In addition to these actions, new icons can easily be defined and added to modify objects in different manners.

7.3.2 Visualization gadgets

Currently, the collection of visualization primitives in the DIVA package comprises three gadgets: the cone tree, the histogram and the graph gadget. This set proved to be relatively complete for the purposes of visualizing business process simulation. Each of these gadgets are discussed in somewhat more detail below.

The cone tree

The cone tree was developed at Xerox Parc and has since then become one of the best known examples of 3D visualization (Robertson et al. 1991). The idea behind the conetree is that the 3D representation of a tree structure makes optimal use of the screen space and thus enables the visualization of much larger structures than the traditional 2D approach (Koike & Yoshihara 1993).

The requirements for our implementation of the cone tree are as follows:

- As the cone tree is meant to visualize large data sets, an important requirement is that the implementation must be efficient. It must be able to display several thousands of nodes without a problem.
- The implementation must be able to display dynamic trees. Changes in the underlying data-structure must be visible in the cone tree.
- The user must be able to navigate through the data set by manipulating the layout of the tree. In the case of the cone tree this is realized by allowing the user to rotate each of the cones.
- Users must be able to select nodes in the tree. Additionally, the gadget must be able to change the constellation of the branches in such a way that the selected path is rotated to the front.

Figure 7.4 shows the visualization of a directory structure of a file system. The visualization continuously monitors modifications of the underlying file structure and adapts the tree when necessary.

Performance was the major design requirement for this gadget. Therefore, we have implemented critical elements of the tree using our own customized graphical objects. For example, instead of using a standard scenegraph consisting of group and transform nodes to create a single cone with associated text planes, we created a special purpose `TreeTextPlane` object that patches translations into its defining vertices. Additionally, all `TreeTextPlane` objects share an alphabet of textures to decrease memory allocation. A drawback of this approach is that adapting the tree to new requirements has become more complex.

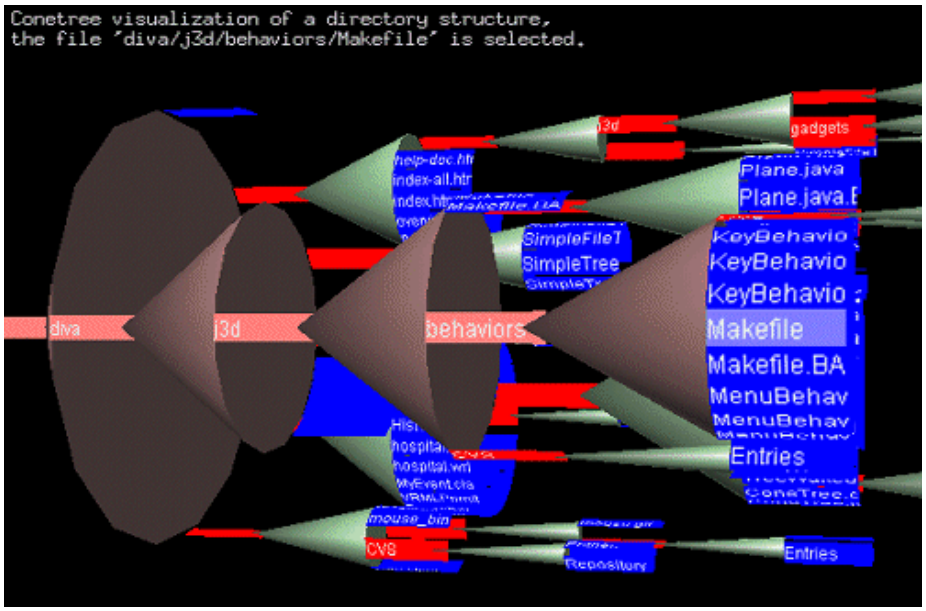


FIGURE 7.4: A cone tree visualizing a directory structure.

The 3D histogram

The histogram is a visualization primitive that is used to give insight into the distribution of data over a specific quantity, such as time. The advantage of a 3D histogram over a 2D histogram is the fact that the third dimension can be used to show additional information. This way, multiple data sets can be compared. For example, the monthly sales results of three competing companies can effectively be compared using 3D histograms.

As requirements for our 3D histogram we state:

- The dimensions of the histogram must be customizable.
- The histogram must be able to handle an arbitrary number of bars of arbitrary size and shape.
- The text displayed at the axis must be customizable.
- The histogram must be able to change dynamically whenever the input values for the visualization change.

Figure 7.5 shows a screenshot of the 3D histogram in action. Because 2D histograms are often used in (business) visualizations, we have designed the gadget to resemble 'normal' 2D histograms. The 3D implementation, however,

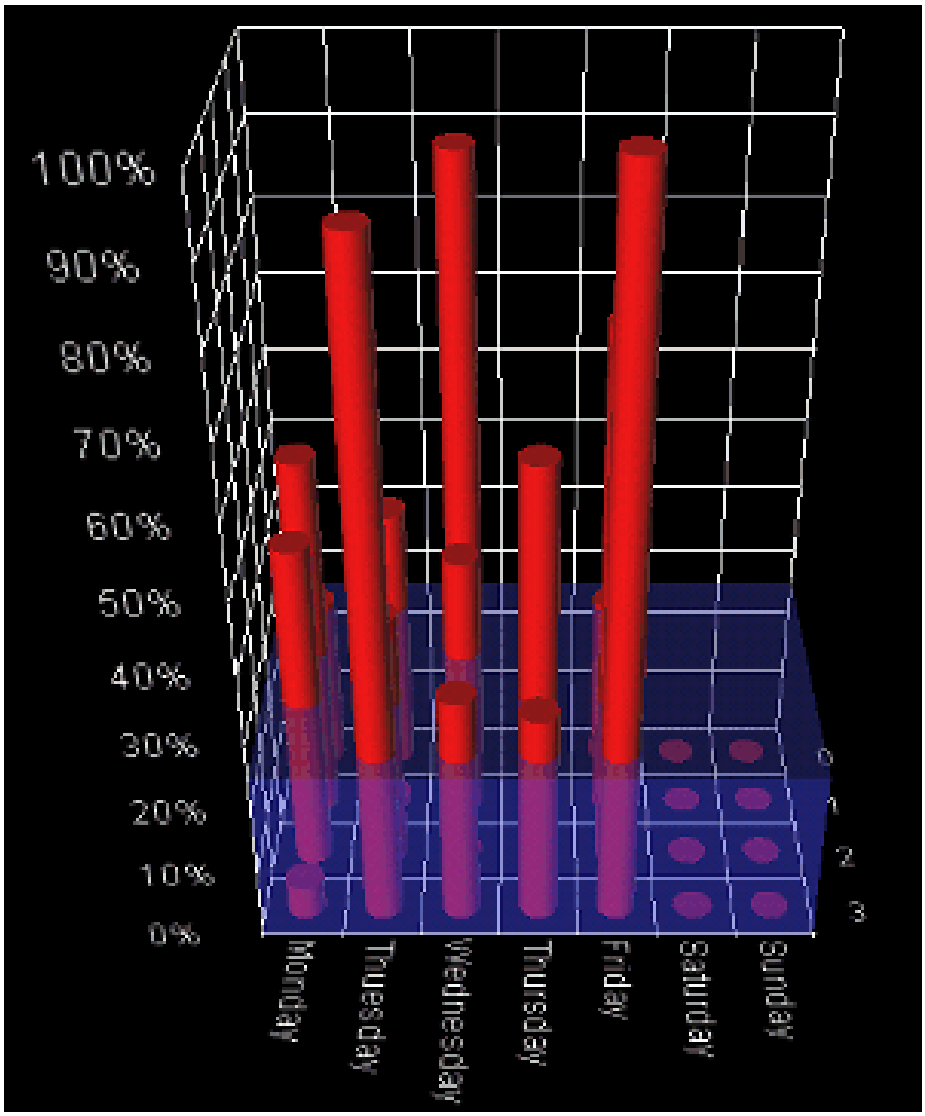


FIGURE 7.5: The histogram gadget.

adds the possibility to show multiple rows of bars at the same time. In addition to this, the gadget has a so-called water-level, which can be used to indicate to the user what level is to be considered normal. The water-level can be seen in Figure 7.5 as the transparent blue box filling the lower 25 percent of the histogram gadget.

An important feature is that the histogram works with generic bars. All that a bar class must do for the histogram gadget to be able to use it is implement

the `Bar` interface. This makes it possible to implement specific bars for specific purposes. There are two standard implementations of the Bar interface included in the DIVA package. The first option is a simple bar, which is nothing more than a red cylinder that can grow and shrink. A second implementation is the multi-level or stacked bar. This class allows for sub bars with customizable colors that can all grow and shrink independently of each other. The histogram gadget uses the brushing behavior to allow users to retrieve additional information about each of the bars.

The graph

The purpose of the Graph gadget is to visualize the structure of and transitions in dynamic (business) processes. It is capable of representing both the static model (a graph) and the dynamic behavior in the form of elements flowing through the model. Requirements for the graph gadget include:

- The gadget must be able to visualize activities within the graph by means of tokens flowing over the edges of the graph.
- The user must be able to change the layout of the graph.
- The user must be able to collapse subgraphs in order to focus on what is important.
- It must be possible to select nodes and interact with the underlying information source (e.g. a simulation).

The graph gadget consists of nodes which are connected by edges. To visualize a flow in the model, tokens move from node to node over the edges of the graph. The BrushingBehavior provides the user with information about nodes and tokens. Manipulation of graph elements is achieved by means of the ModifyBehavior. To allow users to select and (de)iconify groups, two new icons are added to the default modify behavior. One icon allows users to select the parent group that the currently selected item is part of. The other icon allows users to collapse (iconify) or expand (de-iconify) the currently selected group.

Figure 7.6 illustrates usage of the graph gadget to visualize a simulation of a counter-based business process, such as can be found at banks or fast food restaurants. From left to right, the graph structure depicts arrival, a single waiting queue, two counters and departure. People, represented by the small tokens, enter the process on the left and wait in the queue until one of the counters is empty. After being served, tokens leave the process graph again on the right-hand side.

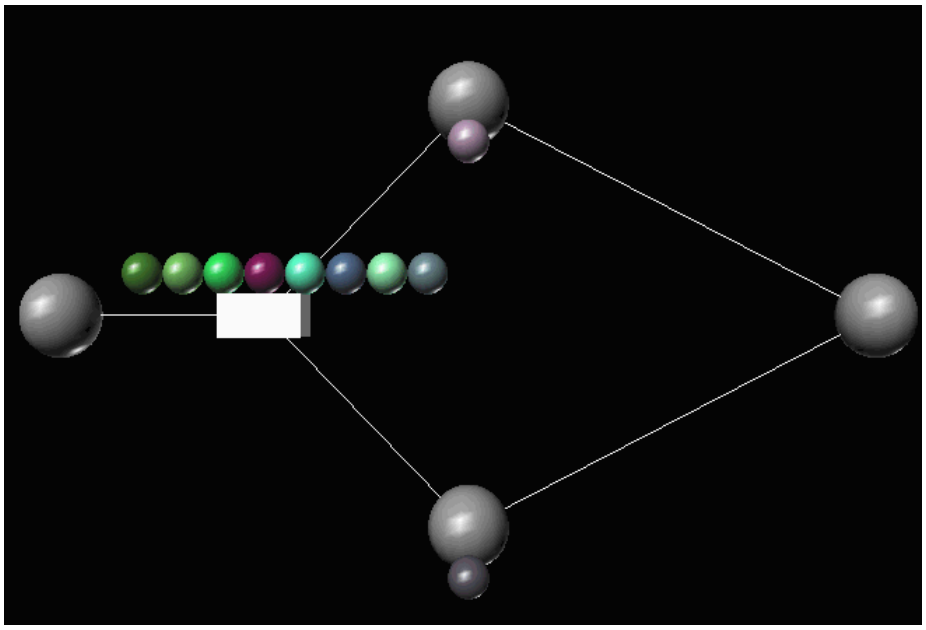


FIGURE 7.6: The graph gadget.

Additional gadgets

The set of gadgets can be extended; more visualization primitives can be developed that would be useful. The results of our work and the experiences gathered indicate that Java3D is well suited for this kind of work. However, our current purposes are not to create an exhaustive set of 3D visualization primitives. Rather, we want to rebuild a 2D visualization application using 3D techniques to evaluate the advantages and disadvantages of the 3D approach to business visualization. The collection of gadgets we have built allows us to experiment with 3D visualization in an actual business process.

7.4 Case study: Visualizing Business Processes

The DIVA gadgets are deployed in a 3D business visualization study at Gak Netherlands. We used the two-dimensional visualization as a starting point since it proved to be successful in the organization during previous evaluation sessions. However, with respect to its 2D counterpart, we extended the 3D visualization with animation and interaction facilities.

As the data source for visualization, our system uses a business process simulation. While the simulation is running, both status and preliminary results of the simulation are visualized by a 3D graph and histograms. This enables

users to inspect the ins and outs of the business process and, when necessary, modify the parameters of the simulation.

7.4.1 Overview

Figure 7.7 is the initial screen of the visualization prototype. It consists of a 3D view, a few control buttons and a chat box. The chat box enables users visualizing the same business process simulation to chat with each other in order to support the process of collaboration.

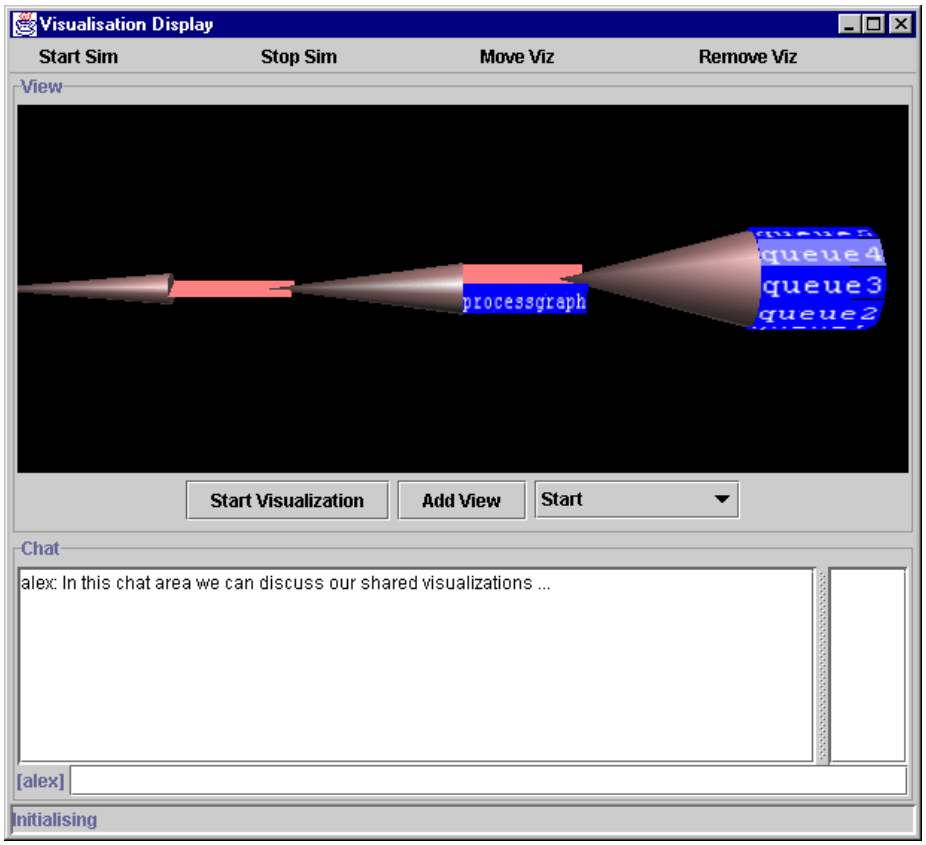


FIGURE 7.7: The initial visualization screen.

Initially, the 3D view depicted in Figure 7.7 only contains a cone tree. The cone tree presents the set of available visualizations: a process graph and a collection of histograms.

Below the 3D view, there are three controls that allow the addition of new visualizations as well as viewpoint navigation. Pressing the *Start Visualization* button results in the creation of the visualization that is currently selected in

the conetree. This allows the user to start any of the available visualizations. The *Add View* button allows users to dynamically add viewpoints to the list of available viewpoints. The control on the right allows users to select a viewpoint from the list of available viewpoints. Once the user selects such a viewpoint the camera moves smoothly to this new location. For each visualization gadget in the 3D view a predefined viewpoint is available. In addition to this, users can add custom viewpoints by the *Add View* button.

The most important visualization in the prototype is the process graph of the simulation. It can be created by selecting *processgraph* in the cone tree and pressing the *Start Visualization* button. The graph gadget visualizes the static structure of the business process, i.e. the stages it consists of. Additionally, the flow of benefit applications through the simulated business process is visualized by means of animated tokens. Figure 7.8 shows the process graph of a business process consisting of 5 processing stages (the large spheres) and 5 queues for applications waiting to be processed (the boxes). The business process visualized in Figure 7.8 is the same as the process used in the 2D version as was shown in Figure 7.1.

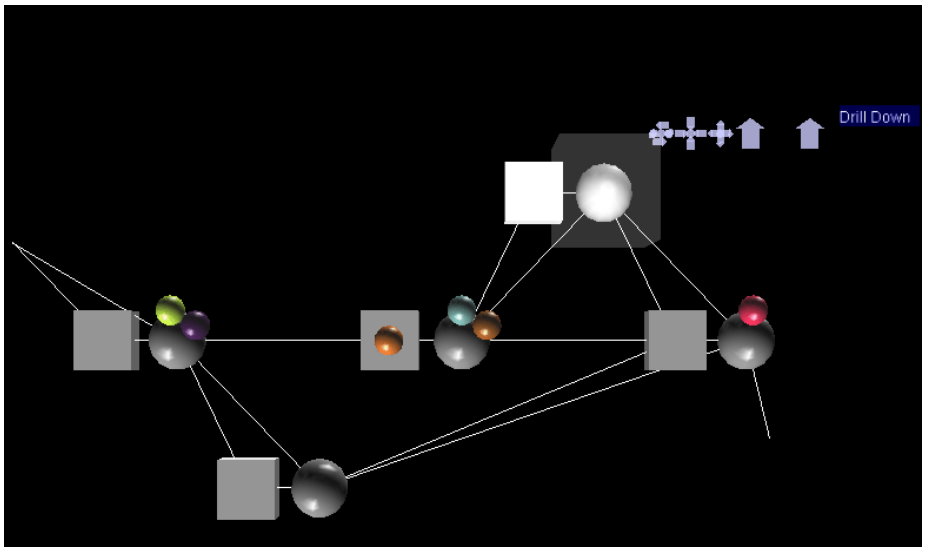


FIGURE 7.8: The process graph of which a single node is selected.

Benefit applications are visualized as small colored spheres. When a benefit application progresses through the simulated process, the sphere depicting the application travels a corresponding trajectory in the process graph.

7.4.2 User interaction

The process graph makes use of the brushing behavior. When the mouse pointer is over one of the nodes (either a queue node or a processing node) or over one of the application tokens, the brushing behavior displays the name of that specific element.

Another means of interaction is provided through the modify behavior. The graph uses the modify behavior to allow users to move, rotate and scale any of the nodes in the graph structure.

As was mentioned in the discussion of the graph gadget, the graph gadget consists of groups of objects. For example, in the prototype the queue and process nodes constituting a single phase are grouped together. In order to manipulate these groups, the graph gadget adds a fourth and fifth icon to the modify behavior. These two additional icons respectively select the parent group of a selected node and collapse or expand a selected group. This allows, for example, a simplification of the graph to five objects representing the stages of the process.

7.4.3 Insight in present and past

The combination of presenting the process structure, the length of waiting queues and the flow of applications through the process gives managers insight into what is going on in a process at a particular moment in time. However, only information about the current situation is not enough to control a business process. Therefore, an important aspect of the prototype is the ability to present summary information of past weeks by means of histograms associated to the queues and process nodes of the graph.

Drilling-down to reveal historical information can be achieved by another icon that is added to the modify behavior. When the drill-down icon is clicked, the histogram associated with the currently selected processing or queue element is displayed. These histograms can also be accessed by use of the cone tree index, but drilling-down on the graph provides a more natural interface to reach these histograms.

As mentioned, both queue and processing nodes of the process graph have histograms associated with them. Figure 7.9 illustrates the histograms that are associated with the processing nodes. These histograms visualize the percentages of applications that are either too early, on time or too late. The x-axis of the histogram is a timeline where each bar represents a single week in the production process. The histogram is intended to give immediate insight into the progress of applications in the admission process.

The histograms associated with the queue nodes visualize the waiting times of applications at a particular stage before they were processed during a (simulated) week. Each bar of the histogram represents a certain time-interval; the

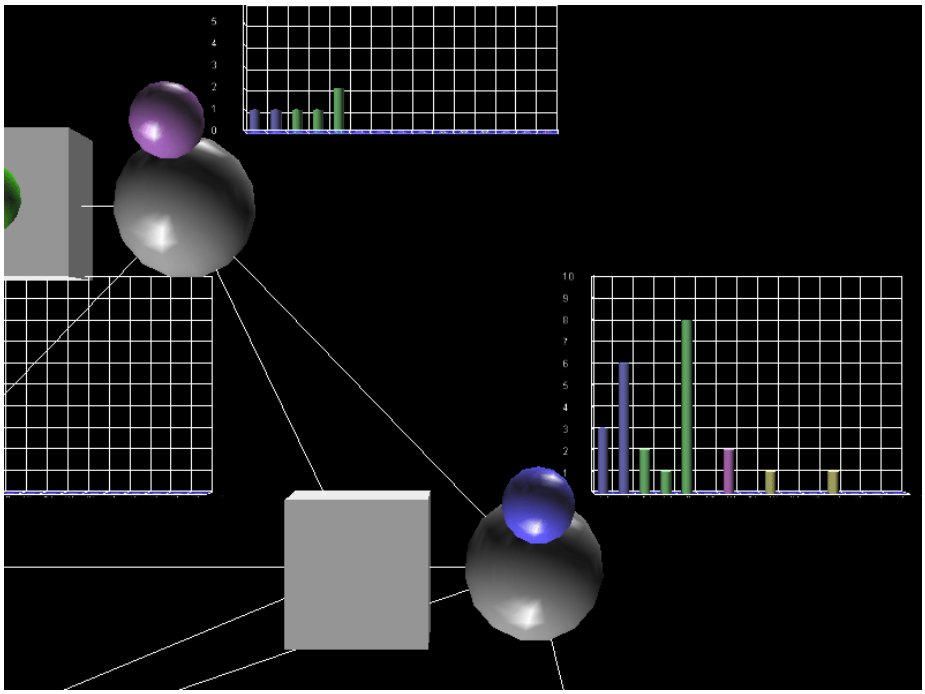


FIGURE 7.9: The drill-down button is pressed and a histogram is created.

height of the bar reflects the number of applications falling within that time-interval. To compare the results of the past week with the weeks before, the histogram contains a 'history' of four weeks of simulated time. The histograms of Figure 7.10 visualize waiting times of applications at all five stages.

7.4.4 3D versus 2D

3D graphics and its application in information visualization applications is a matter of dispute. On the one hand, its advocates promote the usage of 3D because of its close relation with human's three-dimensional intuition. Additionally, 3D visualizations can contain more information at once and are therefore better suitable of presenting multi-dimensional data sets.

Opponents of 3D, on the other hand, state that current input devices fall short in controlling the 3D space and therefore distract users from their primary task (Nielsen 1998). Additionally, 3D applications are often considered as toy applications because they look nice without adding relevant new features.

Experiments with two and three-dimensional visualizations indicate that it is very difficult to compare 2D and 3D directly. A striking phenomenon is the fact that people with more computer experience significantly gain better scores

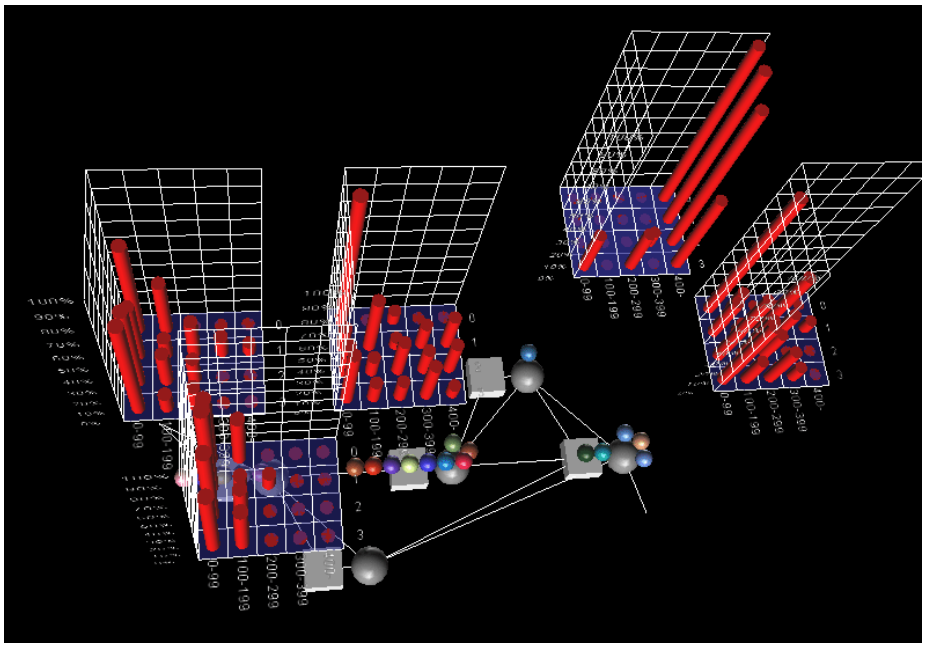


FIGURE 7.10: The process graph and associated queue histograms.

with 3D interfaces than novices. Sebrechts *et al* therefore rightly conclude that *3D visualization cannot be adequately evaluated using only short-term studies of novice users* (Sebrechts, Vasilakis, Miller, Cugini & Laskowski 1999). People will probably have to get acquainted to 3D like they had to get acquainted to graphical interfaces.

At present, we have no empirical data supporting the usefulness of 3D business visualization. The Gak managers, who are the possible future users of the presented visualization system, do not have any experience with 3D business visualization and only limited experience with 2D visualization. However, some qualitative judgements of the strength and weaknesses of the 3D prototype as opposed to the 2D version are presented in Table 7.1.

A disadvantage of the 3D version of the business visualization is the exhibited speed. While the 2D prototype runs smoothly on every desktop PC, the 3D visualizations require a 3D-accelerated high-range PC to reach an acceptable level. A second disadvantage is ease of use with respect to interacting with the visualizations and the underlying data sources. Controlling the 2D prototype is simple and straight-forward. In contrast, in spite of utilities such as predefined viewpoints and manipulation aids, controlling the 3D visualizations is much harder. Another problem to be noted here concerns the acceptance of 3D graphics in business visualizations. Although 2D visualizations are generally accepted as tools to increase the insight in information, the attitude at the Gak

	2D prototype	3D prototype
Speed	Good	Acceptable
Control	Easy	Cumbersome
Acceptance	Good	Tentative
Animation	Limited	Rich
Manipulation	Limited	Flexible
Multiple perspectives	Replacement or multiple windows	Virtual environment, Expand/collapse
Time frame	Single	Multiple

TABLE 7.1: Comparison of the 3D and the 2D prototypes.

company towards 3D visualizations is at best tentative.

On the positive side, the 3D prototype has some interesting aspects not available in its 2D counterpart. For example, the presence of rich animation possibilities in the graph gadget clarifies the dynamics of the business process under inspection. Furthermore, the 3D visualizations exhibit rich manipulation capabilities in the form of the (extended) modify behavior. This allows users to customize the visualization to their personal interests.

Multiple perspectives, i.e. visualizations, on the information space are supported differently by both approaches. The 2D prototype replaces an existing visualization with a new one when a user drills down into the data. Multiple windows are necessary to combine multiple perspectives. The 3D prototype, on the other hand, contains a virtual environment that can contain multiple presentations of the data. Additionally, detailed information can be hidden or shown by means of the *expand and collapse* capabilities of the graph gadget.

As to the question whether 3D visualizations allow for presenting a richer information space, the answer is positive. In particular, whereas the 2D visualization can only represent either historic or present data, the 3D prototype simultaneously visualizes the state of the simulation, the current as well as the historic status of the queues.

7.4.5 Design issues

Although the collection of gadgets is small, it suffices for the purpose of visualizing business process simulation. Since the gadgets are not explicitly tailored to the current situation, they can also be deployed in other business visualization applications. The behavior components are even more generic because they do not depend on the type of information they have to represent. Therefore, the behavior components are reusable in other 3D graphics applications as well.

During the connection of the graph gadget to the simulation we discovered a mismatch between simulations and animated visualizations. Namely, in the simulation events occur when an application has moved from one stage to the next. The transitions are considered as actions that take no time. The graph gadget, however, visualizes transitions of the tokens by animating them as moving spheres between nodes. This movement, of course, takes time. The resulting conceptual clash can only be solved by introducing a visualization that is slightly lagging behind.

7.5 Summary and Conclusions

In the experiment described in this chapter, we have shown the possibility of deploying 3D visualizations in a business context. Based on our (limited) experience with this exploration we have drawn some tentative conclusions regarding 2D versus 3D visualization. In comparison with the 2D predecessor of the prototype we can conclude that the 2D version is more easily accessible. The well-organized 2D visualizations in combination with the intuitive use of colors provide an uncomplicated visualization that is easily accepted by business people. The 3D visualizations on the other hand allow us to combine more information into a single scene. In particular, the process graph may visualize the current status of the simulation while the histograms reveal information about the last couple of simulated weeks. In this case, the 3D visualization offers the possibilities to visualize past, present and simulated data in a single image. We observe, however, that simulation and visualization are not yet generally accepted as instruments for decision making. Moreover, most managers are not familiar with 3D. Yet, what solution is to be preferred is up to the managers who in the end have to make the decisions.

CHAPTER 8

Shared Concept Space

The programmer builds from pure thought-stuff:
concepts and very flexible representations thereof.
The mythical man-month - Fred Brooks.

Chapter 5 introduced the *Distributed Visualization Architecture*. This chapter focuses on an essential element of the DIVA architecture: *the Shared Concept Space (SCS)*, the deployed model for information exchange. The Shared Concept Space decouples the information provider and visualizer by means of a shared data model. Characteristics of the Shared Concept Space are a hierarchical concept structure, support for derived data and dynamic updates.

Structure Section 8.1 introduces the concept of communication mechanisms in distributed applications. Additionally, it describes two communication paradigms. Section 8.2 then introduces the general idea and motivation of the Shared Concept Space. Section 8.3 shows that the Shared Concept Space has its foundations in a couple of design patterns which are described in this section. After that, Section 8.4 discusses the software architecture underlying the Shared Concept Space. Section 8.5 describes some distribution aspects of the concept space. Next, Section 8.6 illustrates the practical application of the SCS in a DIVA prototype. Section 8.7 discusses two experiments we have performed that show the possibilities of a more extended SCS. Finally, Section 8.8 summarizes and concludes this chapter.

8.1 Introduction

Interactive, collaborative visualization applications as targeted by the DIVA software architecture are distributed. **Distributed** in this context means that the application consists of multiple elements spread over a number of machines. Instead of being isolated elements, the components of a distributed application work together in order to achieve a common goal. So, three separate copies of a text editor running at different computers do not make a distributed application. However, when the editors contain the *same* document, allowing for collaborative writing, the whole can be considered as a distributed application.

Distributed applications consist of components and a communication mechanism. The **components** are the functional elements whereas the **communication mechanism** exchanges information and control between the components. The Shared Concept Space, which is the topic of this chapter, facilities the communication in DIVA. Moreover, the Shared Concept Space is a model for data exchange in distributed systems. It discriminates itself from other communication mechanisms by exhibiting some specific characteristics, including derived properties and hierarchical concepts.

8.1.1 Communication paradigms

Distributed systems and applications come in all flavors. Each with different purposes, types of users, and security, scalability and performance requirements. Bank transactions, for example, should be secure and fail-safe, whereas internet shoot-em-up games should be as fast as possible at all costs, even at the expense of correctness.

Since quality requirements, such as performance and scalability, depend on the deployed communication mechanisms between the distributed components, a large quantity of communication technologies exists, each with its own characteristics. To produce some order out of the chaos, we will distinguish between two paradigms: **direct communication through message passing** as illustrated in Figure 8.1(a) and **decoupled communication through shared data** as shown in Figure 8.1(b).

Typically, message passing is being used when a limited number of components (usually two) are requesting services of each other. Information is passed in the form of parameters; instructions are usually communicated by (remote) method invocations. Message passing in a distributed application can therefore best be compared with inter-module procedure calls of single-machine applications.

While message passing supports direct peer-to-peer communication between distributed components, applications based on a shared data architecture have to include an additional tier that contains the data. Each component can have

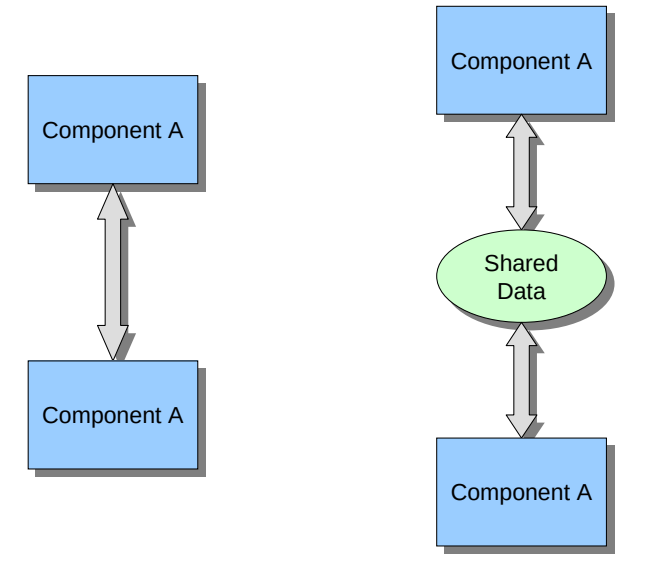


FIGURE 8.1: Two paradigms for communication between components in a distributed application.

access to the shared data independently of other components. Thus, as the name of the paradigm already suggests, the shared data architecture is appropriate to exchange information among a multitude (usually more than two) of components.

8.2 The Shared Concept Space

The communication mechanism of DIVA-based visualization systems is the Shared Concept Space. The main purpose of the Shared Concept Space is to exchange data through a structured data model. The SCS decouples information provider and visualizer components through shared data. It achieves this by providing means to both provider and visualizer to share information about particular data elements: **the shared concepts**.

Whenever an information provider component has new or additional information available, it performs the creation, respectively modification, of a concept. Other components within the application will subsequently become aware of the new piece of information. Thus, the shared concepts allow for sharing data among distributed components. Hence, the SCS conforms to the shared data communication paradigm.

8.2.1 News feed metaphor

The general idea of the Shared Concept Space can easily be clarified by looking at the metaphor of the news feed. In a news feed, news facts are transported from the news source to interested clients. To allow clients to easily identify news facts, the elements are tagged with a descriptive label. For example, the fact that the Nasdaq index dropped 15 points on the 25^{th} of May 2000 could be tagged with the general "`Nasdaq index`" label. This label can consequently be used for all Nasdaq index facts. People who are not interested in stock indexes or Nasdaq at all can thus easily ignore such news facts.

In the Shared Concept Space, news in the form of new data elements and data updates is fed from source to clients. All data elements in the data space are tagged with a label that describes which concept this news fact refers to. This way, information providers have a means to express the topic of the data. In addition, visualizers have a means to decide whether that update might be interesting for their purposes.

8.2.2 Why a Shared Concept Space?

The SCS is not intended as a completely new distributed communication technology. On the contrary, it is built upon existing technology. The Shared Concept Space rather is a new *model* for distributed information exchange. It is explicitly designed for interactive collaborative visualization, but it is certainly also suitable for other types of distributed applications.

Despite the fact that the architecture and implementation of the Shared Concept Space have evolved significantly during the project, the original motivation to introduce a communication model based on shared concepts has not changed. As already discussed in Chapter 5, **decoupling** plays a very important role in the DIVA architecture. The purpose of the Shared Concept Space is to facilitate this decoupling in DIVA.

The scope of the Shared Concept Space, however, reaches beyond distributed visualization only. Other information applications in which components have to share information can also benefit from the SCS. For example, we have created a chat application that deployed the SCS as the communication mechanism between the chatters.

In short, the Shared Concept Space is intended to decouple parts of applications by providing them with a shared data communication model. The added value comes from the possibility to structure the information according to a concept hierarchy. Additionally, the model contains derivation rules to compute derived data.

8.3 Patterns underlying the Shared Concept Space

The architecture of the concept space is built upon, or has a close relation to, a number of patterns. The idea of a pattern to describe a common problem within a certain context stems from Alexander, Ishikawa, Silverstein, Jacobson, Fiksdahl-King & Angel (1977). By means of writing down the problem, constraints and a solution to often occurring practical problems, knowledge which would otherwise remain hidden, can be transferred to less experienced people. Additionally, experts have access to a new vocabulary, the **pattern language**, to express high-level solutions.

Although Alexander's patterns are within the domain of architecture of buildings, the idea of a pattern appeared to be valid in the domain of designing software too. Gamma et al. (1994) define **design patterns** as "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context." Thus, design patterns are best-practice solutions to common problems in the design of (object-oriented) applications.

Architectural patterns are design patterns for software architectures. Buschmann et al. (1996) specify that "an architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them." The purpose of architectural patterns is comparable with the goal of design patterns. They differ, however, with respect to whether they are deployed to architect (high-level) a system or design (parts of) an application.

Below three patterns are given that have significantly influenced the design of the Shared Concept Space. The discussed patterns are Blackboard, Model-View-Controller (MVC), and Talker-Listener. The terminology we use in the discussion below is based on *Pattern-oriented software architecture* by Buschmann et al. (1996).

8.3.1 Blackboard

The idea behind the Blackboard pattern is that a collection of independent specialized components assemble their knowledge by working collaboratively on a common data structure. The Blackboard, named after the real blackboard that human experts use to work together to solve a problem, is the conceptualization of the shared data in the shared data communication paradigm.

Buschmann et al. (1996) define the problem the Blackboard pattern addresses as follows:

> "The Blackboard pattern tackles problems that do not have a feasible deterministic solution for the transformation of raw data into high-level data structures, such as diagrams, tables or English

> phrases. Vision, image recognition, speech recognition and surveillance are examples of domains in which such problems occur. They are characterized by a problem that, when decomposed into subproblems, spans several fields of expertise. The solutions to the partial problems require different representations and paradigms." (Buschmann et al. 1996, p.72)

The Blackboard pattern specifies that the software architecture should comprise several specialized subsystems, each performing its own particular task contributing to the overall process. The blackboard component provides the central data store. The structure of the Blackboard pattern defines a *blackboard* component, a collection of *knowledge sources* (the specialized subsystems) and *control* components which monitor the blackboard and schedule knowledge source activations.

To a certain extent the DIVA architecture can be characterized as conforming to the Blackboard pattern. The specialized subsystems are simulation, databases and visualizers that together create interactive, dynamic visualizations. A commonality of the Blackboard pattern and the concept space is the fact that the components are *independent*, in the sense that they do not directly call each other. In DIVA-terminology, this independence is called *decoupling*.

8.3.2 Model-View-Controller (MVC)

The Model-View-Controller (MVC) pattern is targeted at interactive applications. It divides applications into three components: the **model** encapsulating core data and functionality, the **views** that display the information contained in the model and the **control** components which handle user input by invoking methods on the model.

As part of their problem definition of the Model-View-Controller pattern, Buschmann et al. (1996) state:

> "Different users place conflicting requirements on the user interface. A typist enters information into forms via the keyboard. A manager wants to use the same system mainly by clicking icons and buttons. Consequently, support for several user interface paradigms should be easily incorporated." (Buschmann et al. 1996, p.126)

Whenever the contents of the model have been modified, for example through a control component, all views representing the model are notified. The views can now retrieve the new data from the model and accordingly update the information presented to the user. This is one of the strengths of the MVC pattern: it allows for multiple views of the same model, even when the data model is dynamic.

The purpose of the SCS within DIVA is comparable with the intended use of the MVC pattern. The context of the solution is the same: the data source is dynamic, multiple views or perspectives are required, and both need a means to control the model. A conceptual difference, however, is that the model of MVC represents the functional core of the application including procedures that perform application specific processing. In DIVA, most functionality is located in the processing components, except for the derivation rules, and the Shared Concept Space is more a data communication model than a functional core.

8.3.3 Talker-Listener

The talker-listener pattern, sometimes called publisher-subscriber, is based upon an intermediate communication channel between the talker (subject) and the listeners (observers). The talker publishes its data and information on a channel which takes care of the distribution to the different listeners, as illustrated in Figure 8.2. The advantage for the talker is clear: it does not have to maintain a list of observers. Instead, it can talk to a single interface without having to worry about joining and leaving listeners.

The problem the Talker-Listener pattern addresses is described by Buschmann et al. (1996) (although the pattern is called publisher-subscriber there) as follows:

> "A situation often arises in which data changes in one place, but many other components depend on this data. The classical example is user-interface elements: when some internal data element changes all views that depend on this data have to be updated. We could solve the problem by introducing direct calling dependencies along which to propagate the changes, but this solution is inflexible and not reusable. We are looking for a more general change-propagation mechanism that is applicable in many contexts." (Buschmann et al. 1996, p.339)

Instead of informing viewers that the model has been changed as is the case in the MVC pattern, the talker publishes the changed data immediately. Listeners are able to specify which updates they want to receive through the use of 'tags' like in the news feed. Therefore, listeners do not get disturbed by update messages about information they are not interested in.

In this pattern, the talker is completely independent of the interested listeners. Whether the number of listeners is large or small, and whether it increases or decreases is irrelevant to the involved talker. The bottleneck at the model-side, which might form a scalability problem in MVC pattern, is hereby solved for the talker.

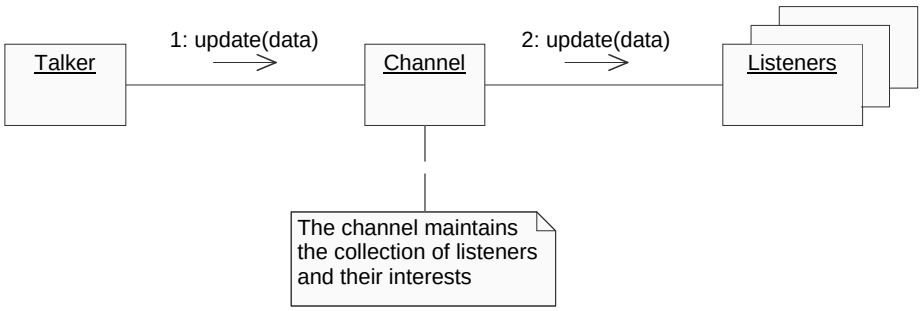


FIGURE 8.2: Talker-Listener pattern

The channel component can deploy a range of strategies to avoid being the single, centralized component as might appear from Figure 8.2. For example, IONA's OrbixTalk[1] distributes information from talker to listener transparently using IP multicast. The channel broadcasts the incoming data to a multitude of listeners by sending it only once to an IP multicast group. The listeners who are part of that IP multicast group pick up the needed data and continue working with the updated information. For the distributed objects, it seems as if they are talking, respectively listening, to single, local objects.

The DIVA architecture decouples information provider and visualizer. The Shared Concept Space, which enables this decoupling, is modeled after the talker-listener pattern. It acts as a collection of channels which are hierarchically ordered. In terms of the SCS, each concept can be seen as a communication channel between an arbitrary number of data providers and visualizers.

8.3.4 SCS and patterns

As a summary of the relation between the Shared Concept Space and the Blackboard, MVC, and Talker-Listener pattern we can conclude that the SCS has picked pieces from all three patterns. The idea of independent but collaborative components that share their information on a central store comes from the Blackboard pattern. The structuring of a system in model and viewer that allows for multiple views comes from the Model-View-Controller pattern. And finally, the distribution aspects via a communication channel come from the Talker-Listener pattern.

[1] OrbixTalk is an implementation of the Corba event service that adds support for scalable and reliable communication. (see: www.iona.com).

8.4 Software Architecture of the SCS

The need for the Shared Concept Space arose from the fact that we needed a powerful information communication model at the heart of the DIVA architecture as shown in Figure 8.3 (reprint of Fig. 5.5). This communication model should support the development of distributed visualizations that enable multiple perspectives on information sources.

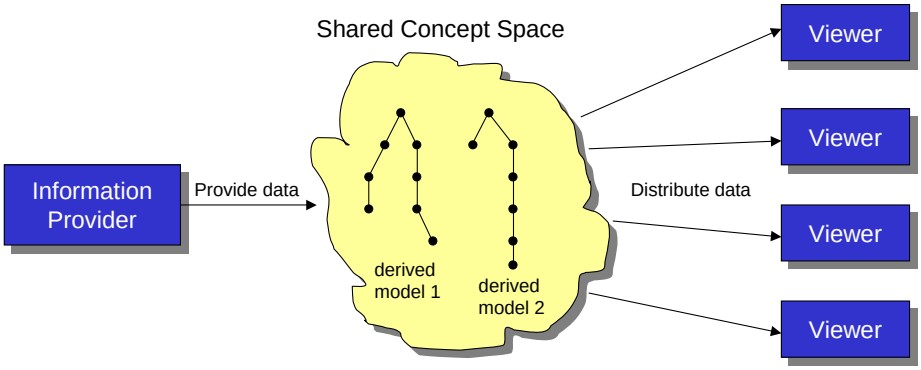


FIGURE 8.3: The Shared Concept Space forms the communication core of the DIVA architecture (reprint of Fig. 5.5).

The most important design considerations for the Shared Concept Space concern hierarchical concepts, derived concepts and dynamic data. This section will discuss each of these issues by going into somewhat more detail of how the SCS works.

8.4.1 Hierarchical concepts

When distributed components deploy shared data, a mechanism to identify and address pieces of the information is necessary. In the Shared Concept Space, we have chosen for a hierarchical collection of concepts since this enables the grouping of concepts even when the information structure is dynamic (see also Section 5.6).

Hierarchical concepts are specified by identifiers separated by dots ('.'), for example `desk1.queue.length`. The last identifier is the name of the concept, the prefix defines its place in the concept hierarchy. Thus, `desk1.queue.length` denotes the `length` concept which is a child of `queue` which again is a child of the `desk1` concept.

To specify what information components are interested in, the `ConceptSpace` interface contains a `register` and `unregister` method. For example, the visualizer of waiting queues has to register its interest by invoking `register("desk1.queue.length")`. The concept space now knows exactly

which listeners are interested in which concepts and can, henceforth, only update interested parties about newly published update events.

In the current SCS, we implemented a number of concrete data storage classes. These classes comprise multiple types, e.g. boolean, integers, strings and floats, as well as multiple dimensions including single objects, sequences and 2-dimensional arrays. An overview of the Data object hierarchy is given in Figure 8.4.

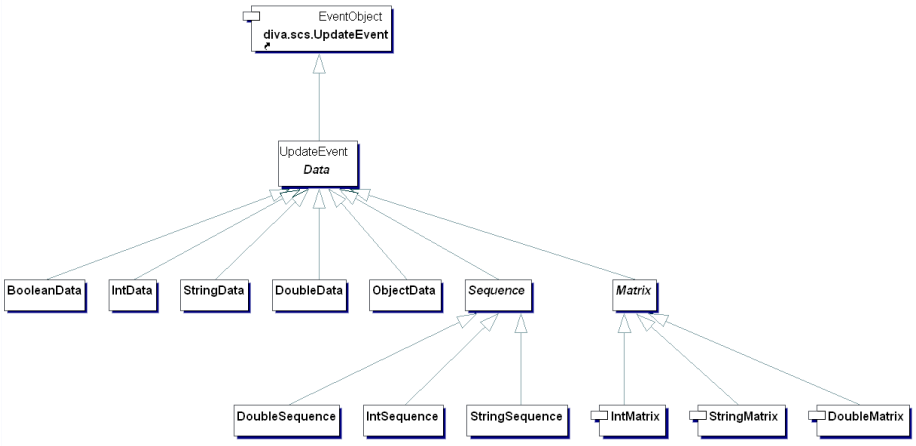


FIGURE 8.4: The object hierarchy of the Data update event.

Since the structure of the concept space is not static, new shared concepts can be added. Therefore, we have extended the means of registration to the concept space by wildcards. Instead of explicitly specifying which update events one wishes to receive, classes of updates can be denoted with a single registration.

Currently, two wildcards[2] are specified: `'*'` and `'<'`. A `'*'` can be replaced by a single identifier. A `'<'` matches any number of identifiers separated by dots. For example both `desk1.*.length` and `desk1.<` will match to `desk1.queue.length`.

The structure of the concept space, including the concept hierarchy, is open. The Shared Concept Space enables all components to create and update new concepts. The communication is thus based on common understanding: the distributed components have to agree on the concept hierarchy.

8.4.2 Derived concepts

To serve decoupled data manipulation in DIVA, the Shared Concept Space contains **Processor** objects. The processor objects complement the data in the con-

[2]We follow the same convention as Objectspace's Voyager.

cept space with derived information. A processor plays the role of data consumer and information provider at the same time. It listens to source data, processes it, and subsequently publishes new information as derived concepts.

As an example, think of a processor that computes the average length of all waiting queues. To achieve this, the processor registers to receive information about `"*.queue.length"`. By summing the individual lengths and dividing the result by the number of queues, the average can be derived. The average length is then published by the processor as the derived concept `"overview.queue.avglength"`.

Processors can be realized in an imperative language, such as Java. However, data manipulation is sometimes a knowledge-intensive task and a declarative language is in those cases a better choice. Therefore, we have included the option to specify derived components using Prolog.

Prolog can be deployed to derive new information based on known facts using rules. This is a similar approach to the derived concepts in the Shared Concept Space. By integrating Prolog into the SCS, we can now use Prolog rules to compute the derived concepts. This enables the combination of Prolog and Java-based derivation processors to cover both symbolic and numeric computation.

8.4.3 Dynamic data

To show how the SCS can serve both static and dynamic data, the necessary interfaces of the software architecture are shown in Figure 8.5. The *data provider* employs the `ConceptSpace` interface to make data or updates available. `Visualizers` implement the `UpdateEventListener` interface and retrieve the `UpdateEvents` containing new data entries or data updates.

An `UpdateEvent`, independent of whether it is a complete or partial update, contains the Concept as its topic. This specifies to which concept in the Shared Concept Space the update refers to. Or in terms of the news feed metaphor, this is the tag to identify the news fact. To publish a new value, the data provider uses the `Data` class. When only relative changes should be published, for example whenever the modification only considers a fraction of the data element, the `Delta` class can be used.

As an example, consider a business process simulation that needs to share the fact that the current waiting queue at desk 1 is 35 people long. To achieve this, the simulation creates an `UpdateEvent` object. In a concept space that is organized according to the physical location of business objects, the topic of the update is something like `desk1.queue.length`. Accordingly, the value 35 denoting the length of the queue is assigned to the newly created `Data` update event.

To share the new information with other interested parties, the simulation calls the `publish` method of the Conceptspace. One of the parties that registered

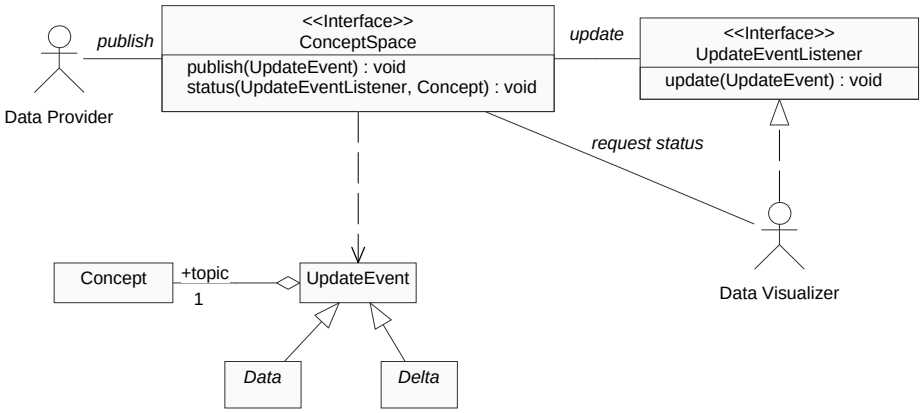


FIGURE 8.5: UpdateEvents are generated by data providers and distributed to visualizers to communicate data and updates.

interest in the waiting queue concept is, for example, a visualization of the variation in the length of wait queues. To receive the update, the visualizer implements the `UpdateEventListener`. As soon as the `UpdateEvent` is distributed over the network, the visualizer receives an `update` method call containing the information as provided by the simulation about the new length of the waiting queue at desk 1. The visualizer can, consequently, update the presented visualization.

8.4.4 Discussion

The most distinguishing aspect of the Shared Concept Space with respect to other communication models and middleware is the possibility to derive information. We will now briefly touch upon the value and possible drawback of positioning this functionality inside the concept space.

An advantage of this approach is the possibility to share knowledge (derivation knowledge) among multiple persons. Derived information, produced by a single processor, can be deployed by multiple users. Additionally, by decoupling data manipulation knowledge from information provider or visualizer, it is possible to reuse this knowledge for multiple purposes. The practical advantage of this form of reuse has already been shown in the 2D and 3D visualization of business processes as discussed in Chapters 3 and 7.

A possible drawback of positioning the derivation knowledge inside the SCS is the additional overhead of creating separate derivation processors. For example, most simulation tools already contain functionality to monitor certain events and present statistical overviews of the occurrence of those events. Moving the functionality out of the simulation into the middleware is difficult and sometimes even impossible. However, when the migration of derivation

knowledge has been accomplished, users can benefit from the knowledge sharing and reuse advantage.

8.5 Distribution aspects of the SCS

When architecting a distributed application, many relevant concerns have to be taken into account. In this section we will briefly touch upon some distribution aspects of our implementation of the SCS.

8.5.1 Distributed system versus single machine

A first performance aspect concerns the distribution overhead. When a user wants to use a DIVA-based system on a stand-alone machine, the distributed nature of the architecture can have a negative influence on performance.

Figure 8.6 shows the approach we have taken in the current implementation of the SCS. Two different implementations of the same `ConceptSpace` interface are defined, each of which can be optimized for its specific environment. This has the advantage that applications deploying the concept space can easily switch from local to distributed concept space or vice versa, as long as they stick to the `ConceptSpace` interface.

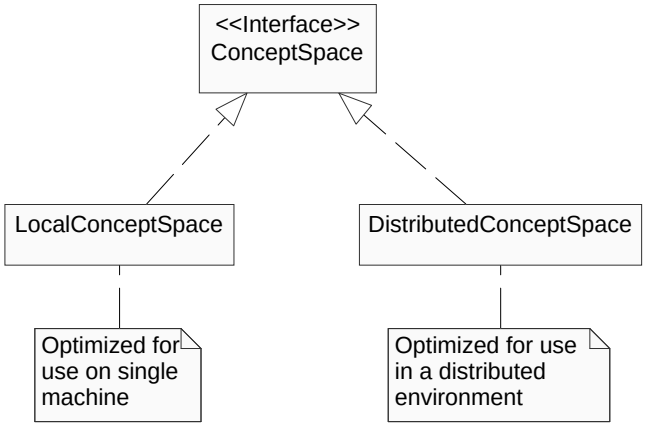


FIGURE 8.6: Two implementations of the ConceptSpace optimize for their local or distributed environment.

8.5.2 Scalability

A second distribution aspect applies mainly to the distributed implementation of the Shared Concept Space. The architecture must be able to scale up to a

larger number of data providers and visualizers. This is definitely in conflict with a *single* Shared Concept Space as it has been drawn in Figure 8.3 since centralized components do not scale very well. For example, Tanenbaum (1995) says about large distributed systems: *... one design principle is clear: avoid centralized components, tables and algorithms (p.30)*.

Therefore, the DIVA software architecture specifies that the distributed implementation of the SCS consists of multiple connected objects that together form a single logical concept space. Each application involved in a visualization session has its own concept space object which is connected to at least one other `DistributedConceptSpace` (DCS) object. The objects, to which a concept space object is connected, are called its **neighbors**. Whenever a data provider calls `update` on its DCS object, the object transfers the `UpdateEvent` to all its neighbors, which again pass the event on to their neighbors. As a consequence, all DCS objects that are directly or indirectly connected to the DCS object of the data provider receive the update event.

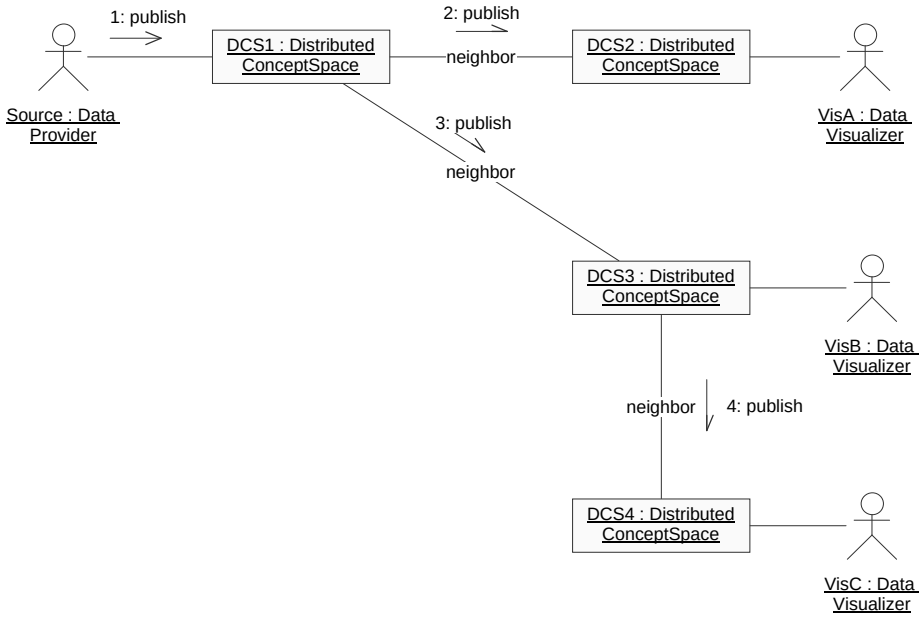


FIGURE 8.7: The distributed concept space consists of multiple connected subspaces that together form a single logical concept space.

Figure 8.7 illustrates the usage of the distributed concept space by means of a UML collaboration diagram. Each data provider and visualizer component is connected to a local DCS object (DCS1 through DCS4). The example given in Figure 8.7 runs on 4 separate machines, each having its own DCS object. The source of the visualization specifies the value of a data element by calling `update` on its local distributed concept space object `DCS1`. To distribute

the newly available data concept, `DCS1` forwards the event to all its neighbors asynchronously. In this case, `DCS2` and `DCS3` receive the data update. Subsequently, `DCS3` forwards the update again, asynchronously, to `DCS4`.

The specification of the `DistributedConceptSpace` avoids centralized components or data tables. New distributed concept space objects can be added to the already existing network of DCS connections without overloading a single component. However, it must be remarked that the architectural solution given here enables a scalable system; it is not guaranteed. How well the resulting systems scale in practice is, amongst others, dependent on the topology of connected DCS objects.

8.5.3 Topology

The topology of the distributed system can influence the network load, and hence the performance, of the distributed system considerably. For example, consider a simulation that produces a large quantity of raw data. The information that a summarizing visualization needs is based on this raw data, but much smaller in size. Exactly for this kind of situations, the SCS contains the possibility to introduce derived concepts. However, the location of the processor component that produces the smaller derived information influences the amount of data that has to be sent over the network. As shown in Figure 8.8, locating the processor object near the simulation decreases the amount of traffic.

More generally, the topology of the Shared Concept Space allows for a good performance of distributed applications. As a rule of thumb for achieving optimal use of network resources, derived concepts producing smaller summaries of large quantities of data should be located near the data provider. Derived concepts that produce more data than they receive as input, however, should be close to their consumers.

8.6 Example usage of the SCS

The Gak business process visualization prototype (Chapter 3) comprises both databases and simulation as data providers. The information visualizers, however, require other, derived data to present the histograms with management information. To produce the derived information, the prototype deploys the Shared Concept Space with a number of Processor objects as illustrated in Figure 8.9.

The contents of the Shared Concept Space are defined by the individual components. The communication is based upon common understanding. A structure enforced by the Shared Concept Space would standardize this communication. However, before the current structure can be formalized into an ontology and

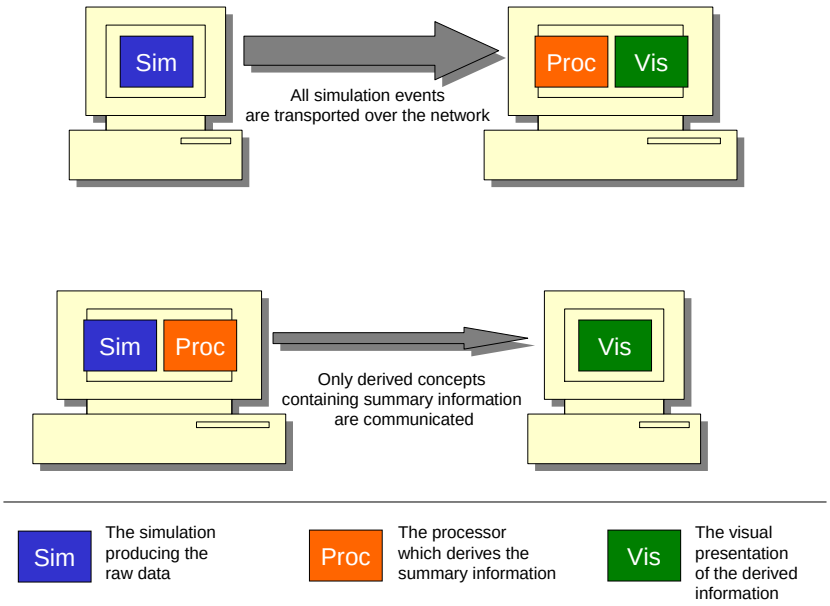


FIGURE 8.8: The topology of the derived information processors in the Shared Concept Space can have a large impact on the network traffic and hence performance of distributed visualization.

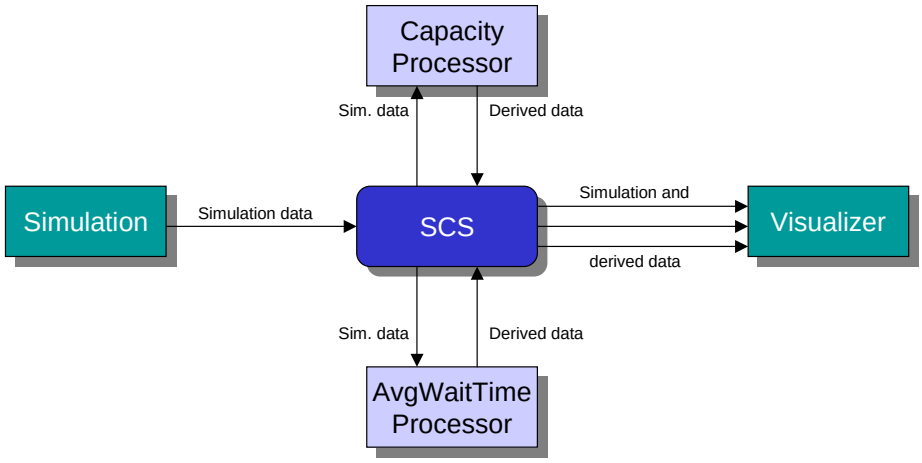


FIGURE 8.9: Processors derive high-level information based on the raw simulation data.

deployed as a general model for business visualization, more research is necessary.

The derived concepts summarize information from database and simulation into sequences which are used as the data source for presenting the histograms. As an example, let us take a look at the `AvgWaitTimeProc` which computes the average waiting time of elements in a particular stage of the business process. To be able to derive this information, the processor monitors the `enqueue` and `dequeue` events of the simulation in association with the current simulation `time`. Based on these facts, the processor can derive individual waiting times of elements in the simulation process.

The AvgWaitTime processor produces overviews of waiting times for each week of simulated time. Therefore, it summarizes individual waiting times into a small list of numbers. The first number represents the quantity of elements that have been waiting between 0 and x minutes. The second number represents waiting times between x and $2x$ minutes, etcetera.

The derived information is published as an `IntSequence` which can easily be presented with a histogram by mapping the individual numbers of the sequence to the height of the bars. Figure 8.10 shows this process by zooming in on the central part of Figure 8.9.

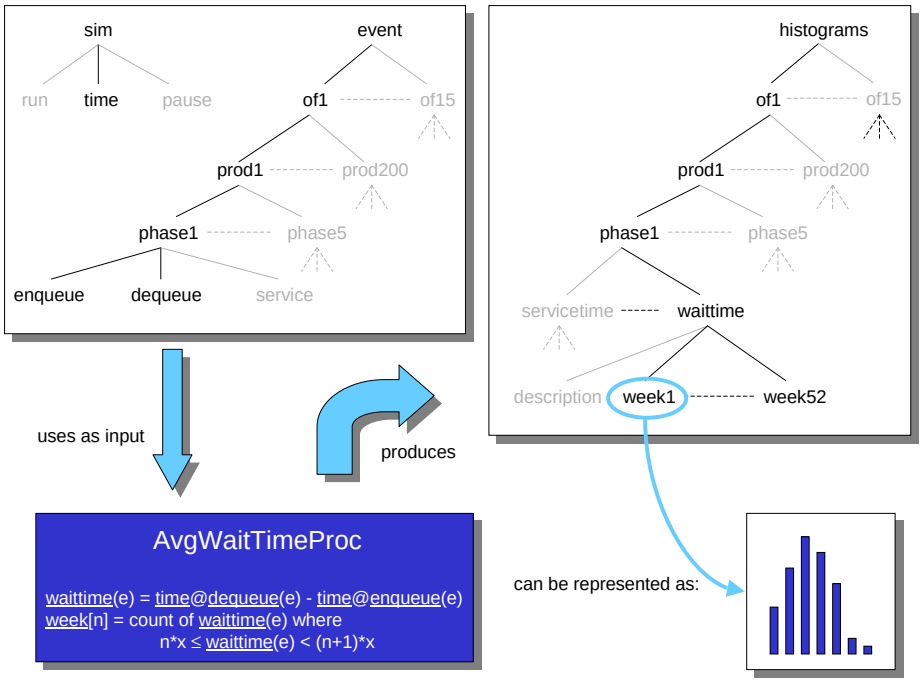


FIGURE 8.10: The AvgWaitTimeProc derives information to produce histograms in the ASZ/Gak business process visualization prototype.

Summarizing, the SCS played an important role in the BizViz prototype, for

a number of reasons. First, the hierarchical concepts enabled the structuring of information from simulation, database and processors in such a way that visualizers could easily access it. Second, the possibility of decoupled data manipulation through derived data allowed us to reuse particular mappings from raw data to high-level overviews for different presentations, e.g. the 2D and 3D visualizations.

Third, the fact that the Shared Concept Space allows dynamic data made it possible to display the intermediate results of simulation. Consequently, the process of simulation, reviewing the results and adapting the parameters has been made more flexible. Finally, the current implementations of the SCS enabled us to create both single-machine and distributed versions of the Gak application.

8.7 Experiments with the SCS

The primary goal of the Shared Concept Space is information exchange. However, through the usage of derived properties and processor objects the SCS can be extended with interesting functionalities. During the DIVA project, we have done a couple of experiments that extend information exchange with *history* and *dynamic query* facilities.

In the **history** experiment, we have created a Shared Concept Space using Prolog that maintained a history of all changes to the data. The processor objects save every update and the time at which it occurred. This extension enables some interesting features for visualization purposes. First, it allows for the recording and playback of simulations, allowing multiple users more freedom to step forward and backward through simulation sessions. Second, the recording of historic facts enables users to search through historical data dynamically by doing queries on the concept space. This approach is more flexible than the approach we have taken in the Gak management information prototype. There, we only recorded historical facts that were needed for our immediate visualization purposes. However, when a new insight would need different historical data, it is already too late and the simulation session has to be restarted since all irrelevant update events are already lost. In case the concept space contains history, no update event is lost and the new view could immediately be created and displayed.

In a different experiment, we have deployed Prolog to **dynamically query** information in the SCS. In this case, every processing step from raw data, which is inserted into the concept space, to the visual model is performed by Prolog processor components in the SCS. Every visual element in the visualization is, therefore, directly connected to a derived concept in the SCS. For example, each bar in a histogram is the visualization of a number that is the result of some derivation performed in the SCS. In a visual manipulation tool, this histogram bar can be dragged onto another visual perspective. The result of this interaction is that the query that resulted in the dragged bar is now the source for the

other visual perspective: we have dynamically created a new query that is the combination of the selection query for the bar in the histogram and the visualization derivation knowledge of the other visual perspective. The selected bar is used as the source for the new visualization. When the capabilities offered by this SCS would be combined with a good and intuitive visual manipulation interface, it would result in a very powerful tool to dynamically visualize and query (dynamic) data sets.

8.8 Summary and Conclusions

The Shared Concept Space is a model for information communication in distributed environments. It is comparable with a news feed that publishes news facts to multiple clients. As a means to organize the information, data elements in the SCS are structured as hierarchical concepts.

The Shared Concept Space is based on the Blackboard, MVC and Talker-Listener pattern, eclectically using elements from all three patterns. The added value of the SCS, however, comes from the possibility to derive new information. This way, derivation and processing knowledge can be shared and reused among multiple components and people.

The Shared Concept Space's original motivation stems from interactive, distributed visualization. It has successfully been applied as the communication model for multiple visualization prototypes. Especially the support for a hierarchical data structure, derived data and dynamic updates appeared to be the strengths of the concept space. Before it can be deployed in business-critical situations, however, more research on restricting the access to concepts and imposing a particular structure on the space is necessary.

CHAPTER 9

Distributed Objects: from Features to Styles

Software development continues to be, as always, a difficult and fascinating mixture of art, science, black magic, and hype.
D'Souza & Wills (1999)

During the last decade, distributed applications have received increasing interest from both users and application developers. There are a number of reasons for this. First, distributed applications have the capability of dynamically sharing data and functionality, allowing users to communicate and collaborate with each other. Second, distributed applications can combine the computing power of multiple computers resulting in very powerful systems.

However, distributed applications are far more difficult to design, implement, test and debug than non-distributed applications. In particular, architects of distributed applications have to make the difficult decision of how to split up the system in order to achieve quality requirements, such as security, performance and maintainability.

In this chapter, we will discuss four architectural patterns or styles[1] to characterize distributed object-oriented software. Additionally, we will discuss

[1] Architectural styles is an often used term for high-level patterns describing specific characteristics of the software architecture of a system. In this chapter we will use the terms *architectural pattern* and *architectural style* interchangeably.

guidelines for their usage, and indicate which technologies can be deployed to implement the styles. Thus, by classifying and cataloging distributed OO software in architectural styles and providing guidelines for their usage, we are trying to give some guidance for the development of distributed software.

Structure Section 9.1 briefly introduces the idea of architectural styles of software. After that, Section 9.2 discusses an object-feature-space to classify objects in distributed OO software. However, since software architects are not so much interested in object features *per se* but more in the behavior of the system as a collection of collaborating objects, Section 9.3 presents four architectural styles.

The **distributed objects**, **dynamically downloaded classes**, **mobile objects** and **event-space** architectural styles are all based on distributed OO technology but differ in supported object features, connector types and location issues. To illustrate the practical usage of styles, Section 9.4 shows how the styles are deployed in DIVA to allow for the addition of new visualization perspectives. As a conclusion, we will evaluate the architectural styles and provide some rules-of-thumb for deploying them in Section 9.5.

9.1 Software Architecture and Style

Software engineers, both academic and in industry, are increasingly interested in Software Architectures. An important aspect of software architecture is the fact that it forces us to think about the quality requirements of a system in addition to the functionality. Moreover, software architecture allows for an evaluation of software systems early in the development cycle.

Architecture-based development promises, amongst others, high-level reuse and separation of concerns. But what we are mainly interested in here is that, while traditional development approaches are primarily concerned with functionality, software architectures are concerned with the interaction and communication of components (Clements 1996).

Whereas software architectures describe or prescribe one particular system, Shaw & Clements (1997) and Shaw & Garlan (1996) classify groups of software architectures in architectural styles. Architectural styles are *descriptions of component types and a pattern of their runtime control and/or data transfer* (Bass et al. 1998). Architectural styles are often, as in our case, based on practical experience. By making design decisions and considerations explicit by means of styles and guidelines, we are able to transfer this important knowledge to other similar software development projects. Books such as (Buschmann et al. 1996) and (Schmidt, Stal, Rohnert & Buschmann 2000) are therefore a valuable source of best-practice information that can help the software architect.

9.2 Distributed Object Feature-space

Before we discuss the architectural styles, we look at the elements constituting a distributed OO system first: the objects. In the following classification we will examine properties of single objects and the connectors between them. Furthermore, the location of functionality and data and the possibility of changing that location is included in the overview. Since the field of distributed objects is very broad, the issues described here are far from being exhaustive. However, we do think that this set of issues consists of some high-level topics that are relevant for distributed OO software.

9.2.1 Objects

The first category or dimension of the object-feature-space concerns properties of the components comprising the system. In a distributed OO system, these components are objects. Objects consist of an interface, the externally visible data and methods, and an implementation of that interface. Remote objects call each others methods via a well defined interface. When an object exposes **meta-information**, remote objects can dynamically determine the object's interface. Consequently, this allows for new objects to be inserted into a running system and deployed by other objects which use the meta-information to derive the object's interface. Usually, the meta-information description is syntactical only and does not have any semantical meaning.

Additionally, the interface or even the functionality of the objects might be **dynamically changeable**—of course, meta-information is necessary to discover such changes. In this case, an existing object is extended with a new feature resulting in an adapted or additional interface and extended behavior. One form of dynamically changeable objects is *dynamic aggregation* (ObjectSpace 1998) which aggregates two objects with a resulting interface that is the union of the original ones.

9.2.2 Connectors

The second feature category concerns the connectors *between* the components. Roughly speaking, the connector can be of two different types: method-based or event-based. A **method-based** connector enables objects to call methods of a public interface. When the call is done on the same machine (or process) the call is **local**. In case the caller and callee of the methods are on different machines we speak of **remote method** calls.

An **event-based** connector passes events. Events range from signals containing no information to full blown event-objects consisting of functions and data. Whereas method-based calls are usually synchronous, event-based mechanisms are mostly asynchronous.

The main task of the connector is to let objects communicate. However, a secondary task can be the translation of that communication to make objects in different technical contexts work together. A connector is **inter-operable** if it can be employed by other objects written in different languages, or running on different platforms.

9.2.3 Location

Location issues concern the location of both data and functionality in the system. Additionally, it encompasses the fact whether objects are mobile or replicated. By **downloadable** we mean that object-code (classes) can be dynamically downloaded to clients where the objects are instantiated. When components are **mobile**, objects migrate taking functionality, data and status along. Thus, mobility implies that the object (instance) is moved *as-is* whereas downloadable objects are newly created objects based on classes available at a server.

Replication means that multiple identical copies of an object exist to improve performance or reliability of the system. Replication is still a research issue and as such not supported by (commercial) middleware solutions.

9.2.4 Feature matrix of distributed object technology

Table 9.1 shows the classification of a number of distributed object technologies according to the above mentioned features. The illustrated technologies are: OMG's CORBA, Objectspace's Voyager, Microsoft's DCOM, and Java with RMI (Remote Method Invocation) by Sun. A technology representing event-based mechanisms is present in the form of the CORBA event-service. Unfortunately, there does not exist technology supporting replication which is mature enough to be included in the table.

TABLE 9.1: A classification of distributed OO technology

	Object		**Connectors**		**Location**	
	Meta	Dynamic changes	Type	Interop.	Downloadable	Mobile
Corba	+	-	remote methods	+	-	-
Voyager	+	+	local methods	-	+	+
DCOM	+	-	remote methods	+	-	-
Java RMI	+	-	remote methods	-	+	-
Event-service	-	-	events	+	-	-

The **Common Object Request Broker Architecture (CORBA)** (Siegel 1996) as defined by the Object Management Group (OMG) allows objects to interoperate by means of a softwarebus: the Object Request Broker (ORB). Because CORBA has been designed to connect distributed applications written

in different languages running on multiple platforms its components are remotely callable and inter-operable. The Dynamic Skeleton Interface (DSI) allows clients to request which methods are available on remote objects. Therefore through DSI, objects can expose information about their interface (meta-information).

An agent-system such as **Voyager**[2](ObjectSpace 1998) provides means for agents (autonomous 'intelligent' mobile objects) to travel around the distributed environment. Hence, mobility is an important feature of agent ORBs. Since agents move to the same machine to communicate, they employ local method calls to exchange information. In addition to mobile objects, Voyager supports dynamic changes of an object. By means of **dynamic aggregation** objects can be extended with new functionality. This feature is very useful in combination with mobility. Namely, developers can now build agents without bothering about the specific mobility issues. When the object is finished it can then be aggregated with a mobile object hereby incorporating all mobility into the new object.

Microsoft has defined a set of technologies to allow for distributed component-based development of applications in the Windows environment (Microsoft 1998) that is comparable to Corba. The **Distributed Component Object Model (DCOM)** is an extended version of COM, which is used to make applications inter-operable. DCOM components are remotely callable and inter-operable (ports of DCOM to other platforms such as Linux or Unix do exist).

Java with Remote Method Invocation (RMI) (Sunsoft 1998) is based on Sun's platform-independent language Java. Java is capable of dynamically downloading classes to clients where new objects can be instantiated. Additionally, Java contains the reflection API which allows remote objects to retrieve meta-information about other objects. RMI extends Java with a means to remotely call objects on different hosts. Although Java is platform independent, RMI is not inter-operable since it can only communicate with components written in Java and not with components written in other languages.

Finally, an event-service such as the **DCOM or Corba event-services** deploy events instead of methods to exchange information between distributed objects. Different implementations of the event-service specifications allow for different quality attributes with respect to multiple recipients of events, reliability of the arrival of the event, and the possibility to retrieve old stored events.

Although Table 9.1 is sufficiently illustrative to make commonalities between object technologies clear, it does not have the right abstraction to classify architectures of distributed object oriented systems. The main reason for this is that it is aimed at features of single objects (the components of the architecture) instead of being aimed at the overall architecture. However, the table illustrates

[2]Note that Voyager also supports remote methods and distributed objects in a CORBA fashion. However to illustrate the differences between an ORB and an agent system we will leave that functionality out of the discussion.

which technologies might be useful to support the implementation of systems based on one of the styles discussed next.

9.3 Architectural Styles

One can find a lot of styles and/or patterns in distributed object-oriented systems. For example, Mowbray & Malveau (1997) contains almost 40 patterns concerning CORBA. These patterns address multiple problems and together form a pattern language that covers both higher and lower-level problems. In contrast, we will only present four architectural styles. These styles are very high-level patterns which address the same problem. The main problem solved by our architectural styles is to communicate and exchange information between multiple objects in a distributed environment.

9.3.1 Distributed objects architectural style

The first style, the distributed objects architectural style comprises software architectures which consist of software components providing services to client applications or other service components. Figure 9.1(a) schematically shows the distributed objects style. Each object is located at a single, fixed place. Objects on different machines are being connected by proxy objects and an Object Request Broker (ORB) that abstracts from the used network and programming language.

The constituent objects are remotely callable. Because distributed objects only expose their interface via an ORB they are inter-operable. Example technologies supporting this architectural style are CORBA and DCOM.

Example One of the standard examples to illustrate distributed objects architectures is a banking application. Such a system typically exists of a single object representing the bank and a number of objects representing the clients of the bank. To withdraw money from the bank, a client component (which is hosted on the clients' machine) calls the `withdraw` function on the local proxy that represents the bank object. The proxy object, consequently, communicates with a remote proxy to invoke the same method on the bank object and to return the result of the transaction to the client object. From a more abstract point of view, the client component has now called the `withdraw` method on the remote bank component.

9.3.2 Dynamically downloaded classes architectural style

The second architectural style is the dynamically downloaded classes style. Here classes are downloaded to and run on client machines, as illustrated in

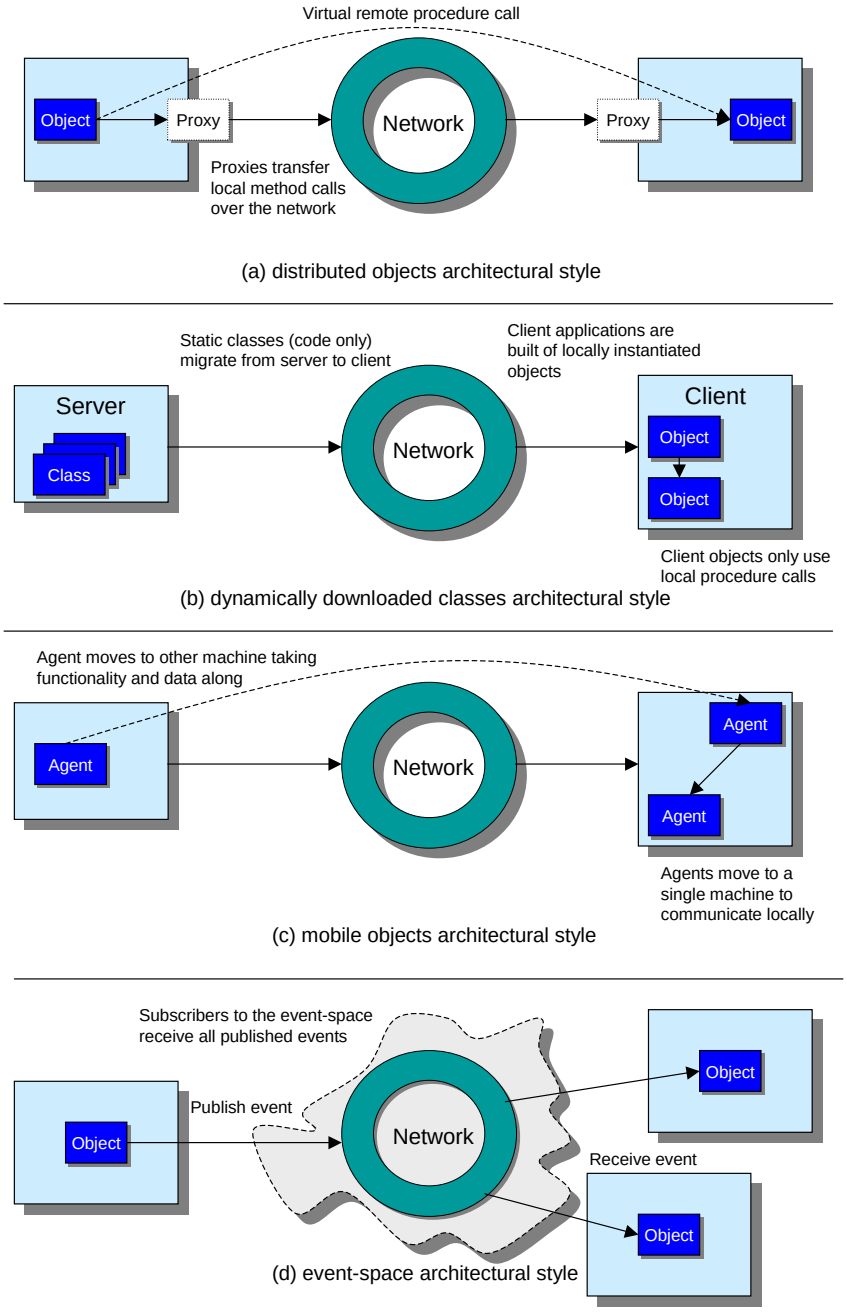


FIGURE 9.1: Architectural styles for distributed OO software

Figure 9.1(b). The dissimilarity between distributed objects and downloaded classes architectures is that while functionality is fixed at one place in the former, it is transported when needed in the latter. Objects, which are running on client-machines, are instantiated from classes which are dynamically downloaded from a server. This implies that client applications are always using the latest version of the available classes, alleviating the job of maintaining a large number of client applications.

In terms of the object-feature space of Section 9.2, the objects in this style are downloadable. Additionally, they have to contain meta-information because the client application has to know how to use the newly downloaded object. Example technologies supporting this style are Java applets, JavaBeans and ActiveX.

Example A lot of examples illustrating the downloaded classes architectural style can be found on the World Wide Web. Small applications (applets) written in Java are downloaded to clients' browsers where they are executed. Thus, the functionality of the Web browser is extended with the functionality of the downloaded applet. Data needed to run the applet is retrieved using the browser's Internet access or Java's specific networking capabilities.

9.3.3 Mobile objects architectural style

A third style is the mobile objects architectural style. Mobile objects migrate from host to host, taking both functionality and data while they move, as illustrated in Figure 9.1(c). Consequently, mobile objects communicate with local objects at the host they currently reside on. This means that mobile objects are a perfect means to implement agents, which wander through a network while collecting information, negotiating with other agents and reporting back to the user who launched the agent.

Mobile objects are, obviously, mobile and often remotely callable, e.g. to invite an agent to your machine. Technologies supporting the mobile objects architectural style are agent ORBs such as Voyager (ObjectSpace 1998) and Mole (Baumann et al. 1998).

Example As an example of the mobile objects architectural style, consider a shopping agent which strolls over a network looking for bargains. The agent moves from host to host communicating with a number of selling agents. After the agent has found something, it returns to the user who sent away the agent reporting the result of the search. The user now decides whether she buys the product based on the information gathered by the agent. This example illustrates that the agent has to keep status to report back to its user. Additionally, network overhead is kept minimal because all negotiations between client agent and selling agent take place locally on the server's machine.

9.3.4 Event-space architectural style

The fourth and last style discussed here is the event-space architectural style, which is illustrated in Figure 9.1(d). In this style an event-space plays the central role of communication channel. Objects communicate with each other by causing events on one or more recipients. Because all objects are connected to the same event-space, broadcasting events to all other members is relatively easy and well scalable.

In terms of the features, event-spaces only require an event-based connector. Technologies supporting event spaces are amongst others the Corba event-service and Voyager's Space.

Example A typical example for the event-space architectural style is a system that distributes stock-exchange information in real-time. As soon as new information is available, the clients of the service receive an event containing the new value. The distribution of the events to all clients is achieved through the event-space.

9.4 Styles in Diva

To illustrate the architectural styles described above, we will now show how the styles are deployed in DIVA. Therefore, we will focus on one particular aspect of the architecture to illustrate the styles, namely, extending the functionality of the system.

The system's goal is to support multiple users in visualizing shared information. During a visualization session, users can come up with new visualization primitives which show the information from a different perspective. For example, one of the users discovers an elegant way to display relevant information that otherwise remains hidden. This new perspective must then be shared with other users to support collaboration between the participants. The remainder of this section shows how the first three styles solve the problem of adding functionality (a new visualization) to the system.

The event-space architectural style on its own is not appropriate to extend a system with new functionality. Therefore, the new perspective example cannot be used to illustrate the event-space. However, an example usage of the event-space architectural style is the Shared Concept Space that was discussed in Chapters 5 and 8.

9.4.1 Distributed objects

If our system would be based on the distributed objects architectural style, the only way we can extend the functionality is by adding a new distributed ob-

ject to the runtime system. Figure 9.2(a) illustrates the process of adding a new visualization perspective. On the left we see a user who has created a new visualization which is made available by means of a distributed object; on the right hand side is one of the users who wants to take a look at the new visualization perspective. At the bottom of the image we see the shared information which is updated periodically.

Once somebody announces that a new visualization primitive is available, other users can connect to the distributed object and request that visualization of information (1). Consequently, the object retrieves the information from the shared information server and creates a new visualization (2). Finally, the resulting visualization is sent to the requesting client (3). Whenever the information is updated, visualizations will be updated by the distributed object and transmitted to all users (`update`).

9.4.2 Downloaded classes

When we base the visualization application on an architecture that conforms to the dynamically downloaded classes architectural style, users who have created the new visualization perspective provide a class that can be downloaded to other users' machines. As Figure 9.2(b) illustrates, a user connects to the server that contains the class of the new perspective (1). This server can be the user's machine, or a shared server to which the class has been uploaded. After that, the class is downloaded to the client's machine and instantiated as a new visualization object in the local visualization application (2). Finally, the information is retrieved from the shared information server and accordingly visualized (3). Because each user contains the functionality to present information at her own machine, updates have to be sent to all client machines.

9.4.3 Mobile objects

The third architectural style, based on mobile objects, is in some respect similar to the dynamically downloaded classes architectural style. In both cases functionality —*how* we visualize information— is downloaded from a server onto the client machine. However, a mobile object keeps its current status. In Figure 9.2(c), we see that a user requests an agent from the user with the new perspective (1). Consequently, the display agent clones itself and moves the clone to the requesting user's machine (2). The clone, which contains all the knowledge the originating agent has, does not have to contact the shared information server to show the visualization on the user's display. Updates of the shared information have to be sent directly to all clones of the original display agent.

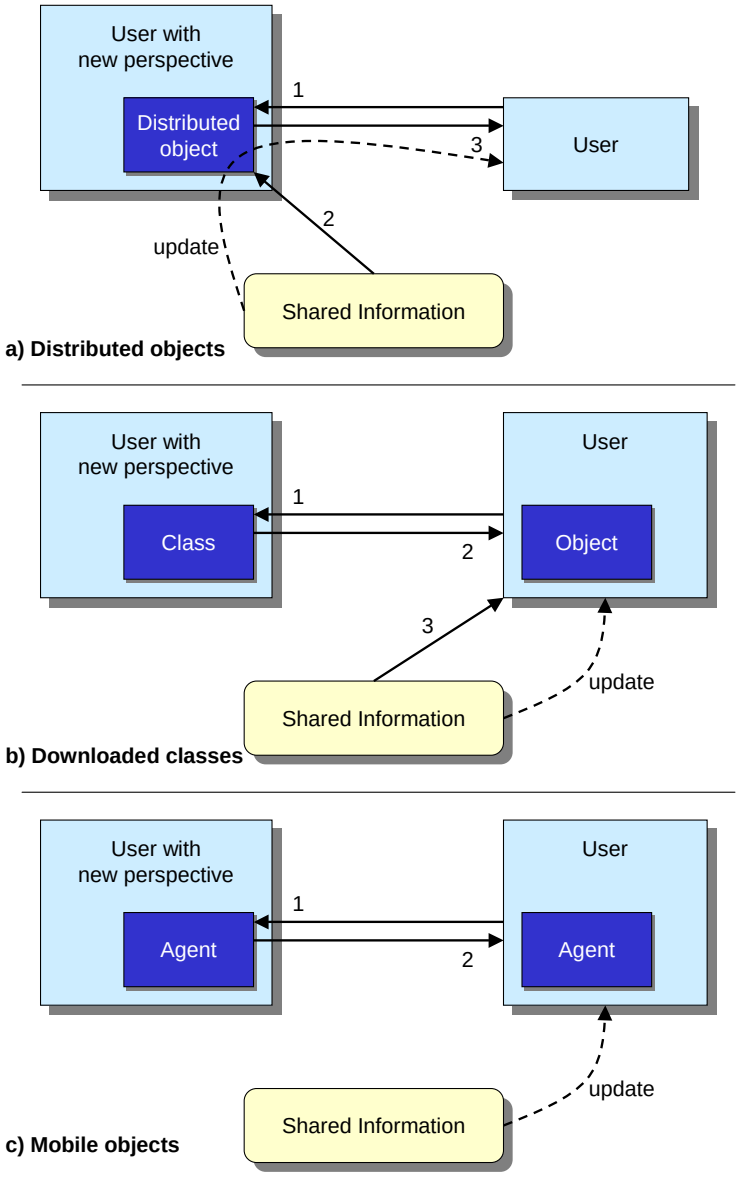


FIGURE 9.2: Architectural styles in Diva

9.5 Evaluation and Discussion

The goal of describing and classifying the four architectural styles in the previous section is to provide a basis for deciding on the right type of architecture when creating a distributed object oriented system. In this section we will ex-

amine features of the discussed styles and, additionally, give some rules of thumb for choosing which style to deploy in particular cases.

9.5.1 Feature-based classification

Table 9.2 contains a feature-based classification of the architectural styles discussed. The **constituent parts** describe the primary building blocks of architectures: components and connectors. The **communication issues** tell us something about the characteristics of the communication that takes place between the components in a style. The **functionality issues** determine whether the functionality of the overall system or parts of the system is fixed or extensible.

TABLE 9.2: Feature classification

	Constituent parts		Communication		Functionality issues		
Style	**Components**	**Connectors**	**Model**	**Topology**	**System**	**Client**	**Server**
Distributed objects	object	ORB	pull	1-1	extensible	fixed	extensible
Dynamically downloaded classes	classes/objects	various	pull	1-1	extensible	extensible	fixed
Mobile objects	objects/agents	procedure call	push / pull	1-1	extensible	extensible	extensible
Event-space	objects	events	push	1-n	fixed	fixed	fixed

In Table 9.2, the communication in a style is characterized by two features: communication model and topology. The model specifies whether the objects use a pull or a push model of information exchange. In case of the pull model, the client asks a server object for information and thus pulls the information from the server to itself. In contrast, when the server sends the information to clients without a previous request from the client, the communication is based on a push model.

The topology specifies the number of objects involved in a single communication event. Thus when two objects are directly communicating, the topology is 1-1. When one server talks with multiple clients at the same time, we speak of 1-n. As shown in the table, only the event-space style exhibits a multiple client (1-n) communication topology.

The functionality issues state that software based on one of the first three styles is extensible at run-time: new functionality can be added without recompiling or restarting the system. The distinction between the styles, however, is the location of this extensibility. Systems based on distributed objects architectures extend the functionality of systems at the server side, while dynamically downloaded architectures achieve this by extending the client components. In the mobile objects architectural style, objects can migrate to other hosts which implies that both client and server parts are extensible.

9.5.2 Rules of thumb

Table 9.3 contains some rules of thumb about when to use a particular style. We state that a thorough understanding of the interplay between the deployed object technology, architectural styles and the software architecture of a particular system can aid the architect in designing a system that will meet the quality requirements such as performance and scalability.

Distributed objects are often regarded as the object oriented variant of client–server. However, distributed objects are more than that: distributed objects are namely both client and server at once. Rules 1 and 2 illustrate when it is useful to deploy the distributed objects architectural style. Because inter-operability is a key feature of distributed objects, this style allows the wrapping of dedicated hardware and legacy software into a heterogeneous distributed system (1). Additionally, distributed objects only expose the interface and do not give away the implementation. This allows for its usage in systems where software is not allowed to 'leave' a server because of strategic or security reasons (2).

TABLE 9.3: Rules of thumb

Nr	When to use	Style
1	Dedicated hardware or legacy code	Distributed objects
2	Strategic or secret code (you do not trust to give away)	Distributed objects
3	Lots of users expected, resulting in overloaded servers	Dyn. downloaded classes
4	Often new versions of software are released (maintainability)	Dyn. downloaded classes
5	A lot of communication and.or negotiation between the components	Mobile objects
6	A single source that contains relevant information for a lot of other objects (information-push)	Event-space

When a large amount of clients is expected to be running applications on a single server, the server can easily become overloaded —imagine what would happen when all Java applets would be running on the server instead of on the client's machine. In this case moving the processing to the client, by deploying dynamically downloaded classes, is the natural solution (3). Additionally, when (parts of) applications are updated often, for example because of changing legislation, architectures based on dynamically downloaded classes are much easier to keep up to date. Clients are automatically using the latest version of the available software (4).

Rules 3 and 4 for dynamically downloaded classes also hold for the mobile objects architectural style, because in this style functionality is downloaded to the client's machine too. However, when multiple objects have to negotiate, for example to vote on something, the amount of communication between objects can be very high whereas the result is only a small answer: 'yes' or 'no.' When the communication can be done locally by moving all mobile objects to the same host, the performance of the system can be improved dramatically (5).

A system such as a stock-exchange information system which constantly has to publish its information to a huge number of listeners is a good example to deploy the event-space architectural style (6). Whenever new information is available, the server broadcasts an update event to all interested parties. The central event-space handles the distribution of the events.

9.6 Summary and Conclusions

This chapter started with the presentation of an object-feature-space to classify object technologies. The features, such as connector-type and location, appeared to be useful to characterize properties of single objects. Additionally, we observed that they determine to a large extent the architectural styles as presented in this chapter.

Based on our experiences in developing (distributed) OO architectures we have discussed four architectural styles for distributed object oriented systems: the distributed objects, dynamically downloaded classes, mobile objects and event-space architectural style. Not surprisingly, we discovered that no best style exists. However, particular styles are more suitable to meet specific requirements than others. The discussed *rules of thumb* are a first direction into deciding which style to deploy in particular situations.

CHAPTER 10

Conclusions

A research project is never finished ...
... it just runs out of money and time.

Although it is not the last chapter of the dissertation, this chapter concludes the research described in my PhD-thesis. This is because the next, and last, chapter describes a possible future scenario of how visualization might be used in the *not-too-far-away* future. It illustrates how business managers may control their business processes in the form of a short story. The last chapter is not based upon scientific research but merely inspired by novels written by science fiction authors such as Neil Stephenson and Eric L. Harry.

However, before the *odd* chapter starts, we first conclude the work described in the preceding chapters. After a short summary of all material, this chapter discusses how DIVA contributed to solving the problems discussed in the introduction (Chapter 1). After that, we conclude the thesis by discussing open issues and possible future research directions.

10.1 Summary

Information visualization is the presentation of information through images. But visualization is more than the mere visual display of data. Information

visualization is deployed for a purpose. One goal of using visualization is to understand data, or as Card et al. (1999) state it: *to amplify cognition*. A second purpose of visualization is to communicate information using human's visual, high-bandwidth, capabilities.

Computer-support for visualization used to be directed towards scientific visualization. However, during the last couple of years, information visualization has gained increasing interest. This has resulted in new tools specifically aimed at information visualization as well as built-in visualization support in existing data management tools.

Problems with the current generation of information visualization support tools are threefold. First, most of the tools are aimed at a single user whereas visualization, especially when aimed at problem solving or decision making, is a multi-user activity. Second, many tools have a built-in facility for visualizing native data. However, they do not allow the incorporation of data from other sources. Finally, interaction in visualization environments is still in its infancy. Usually it is possible to interact with the generated visual model, however, interactively modifying the data source or visual mapping is absent.

A specific domain in the realm of information visualization is business visualization or in short BizViz. The goal of BizViz is to support decision-makers with high quality, easily accessible information. In a case study at Asz / Gak Netherlands we have shown that BizViz can help managers in understanding complex information and, subsequently, making better decisions.

Visualization models depict visualization as a process which converts data into images. The visualization process consists of a series of transformation steps, which differ within each different theoretical model. For example, the visualization reference model splits the visualization process in four types of data: raw data, data tables, visual structures, and views. The transitions between those four data types together constitute the mapping from raw data to visual entity. Human interaction, which is explicitly present in the reference model, is allowed during any of the three transitions between the data types.

The Distributed Visualization Architecture (DIVA) describes a visualization architecture which aims at interactive, multi-user visualization in a distributed environment. Hence, it addresses the problems of present-day visualization tools. Additionally, it extends the visualization reference model to the realm of multiple users and multiple perspectives.

The Distributed Visualization Architecture is described according to three architectural perspectives: conceptual architecture, software architecture and information architecture. The most prominent view, the software architecture of DIVA, describes the main components constituting the architecture. Additionally, the software architecture describes the relationships and interactions between the components.

Basically, the DIVA architecture consists of an information provider, a decoupled data model and presentation components. The decoupling of generating

the information, processing it and, finally, presenting it gives the DIVA architecture the flexibility it needs to meet the multi-user requirements.

In the DIVA project, the practical side of software engineering played an important role. Therefore, four case studies have been performed, each focusing on a particular set of problems. For example, *The Great Dictator* was a study towards a collaborative visualization environment. The resulting prototype is based on the basic DIVA software architecture but also comprises collaboration components to facilitate the cooperation of participants. The technology deployed to create the prototype builds upon distributed and mobile objects. Visualizations are presented in the 3D modeling language VRML.

A different technology for presenting 3D visualizations is Java3D. Java3D has been used in the 3D-remake of the 2D visualization of Gak business process information. A small, reusable collection of 3D gadgets was developed that could easily be 'plugged' into the already existing system to experiment with a 3D visualization of management information. An extensive evaluation of the 3D visualizations was impossible. However, it appeared that the most important distinction between the 2D and 3D version is that the 2D visualization is more easily accessible whereas the 3D visualization could contain more information at the price of increased complexity.

DIVA is a distributed architecture. The last part of the thesis is therefore devoted to distributed software architectures. More precisely, the last part focuses on information exchange in distributed environments. In DIVA, the Shared Concept Space (SCS) is the deployed model for information exchange between information sources and visualization components. The SCS is based upon three well-known design patterns: Blackboard, Model-View-Controller and Talker-Listener, eclectically taking parts from all three patterns. The Shared Concept Space, however, has the possibility of deriving new information independent of information provider or information viewers. This way, knowledge for derivation can be shared and reused among multiple components and people.

Architectural styles are patterns describing component types, runtime control and data transfer on the software architecture level. Based on our experience with DIVA we have introduced four architectural styles to classify distributed object-oriented software: distributed objects, dynamically downloaded classes, mobile objects and event-space architectural styles. The styles differ in object features, connector types and location issues. Consequently, they can appropriately be deployed in different contexts depending on the requirements of the architecture. The discussed rules of thumb are a first step towards deciding which style to deploy in particular situations.

10.2 Contributions

In Chapter 1 of this thesis we started with problems of computer-support in the realm of information visualization. These problems formed the point of departure for our trip through architectural perspectives on information visualization. In short, we described three problems with present-day visualization tools:

- **Lack of multi-user support** — Visualization software is mainly targeted at single users while visualization is often deployed in a multi-user activity.
- **Tight coupling** — Existing tools are adding support for visualization. However, most tools integrate the visualization capabilities hard into the existing application leaving no room to include data from other information sources. Combining these built-in visualizations with other built-in visualizations is therefore impossible or at least cumbersome.
- **Limited interaction** — Interaction is mostly limited to the resulting image. Full interaction, however, would certainly improve the user's understanding of the information.

The DIVA project intended to at least partially solve these problems. Now the time has come to take a look at how the DIVA architecture may provide a solution to those problems.

10.2.1 Multi-user

Visualization itself can effectively be used to support collaborative processes such as decision making. This is due to the characteristic of visual information that it is easily transferable. Visualization software, however, is mostly aimed at single users.

The DIVA architecture was designed with the requirement of multi-users in mind from the beginning. It decouples information source, a shared workspace and information presentation, in order to create an effective environment where users can work together. As discovered in case studies and experiments with collaborative visualization, multiple, exchangeable perspectives as supported by DIVA are important in multi-user visualization.

The claim of calling DIVA a full collaborative visualization architecture, however, is too much. Important elements of collaborative environments, such as asynchronously working together or direct communication mechanisms, are not incorporated in the architecture. Summarizing, DIVA directly supports multi-user visualization and provides a flexible foundation which can be extended to support collaborative visualizations.

10.2.2 Coupling

The second issue of computer support for visualization concerns the tight coupling of visualization in information management tools such as databases, spreadsheets and simulation tools. Those tools are built upon an architecture that tightly couples the information source to the data manipulation and visualization functionality. The concept of DIVA is to decouple these functionalities through a shared data model.

The information exchange component of DIVA, the shared concept space, decouples data source and visualization. Different information sources can supply the shared data space with information which can be used by different information processors or visualizers. A successful combination of historic database information and dynamically generated simulation data has been shown in the Asz / Gak management information case study.

10.2.3 Interaction

One of the most difficult problems of computer-based visualization support is the ability to interact with every aspect of the visualization process. In the DIVA project we have shown that it is possible to interact with the information generator (e.g. a simulation), the visual mapping (e.g. by adding new ways of visualizing data) and the final view (e.g. by navigating through the 3D virtual world).

In Chapter 1 we compared interaction in visualization with interaction in the direct-manipulation user-interface paradigm. A strong point of the direct manipulation paradigm is that interaction is so tightly integrated into the visual presentation. Unfortunately, we did not achieve this level of integration in DIVA. Although we can interact with the different components constituting the architecture, each component provides its own means of interaction. A consistent and integrated manner of interacting with the full visualization process has not been achieved.

10.3 Open Issues and Future Research

Near the end of this thesis, it is time to see to what extent the DIVA project is finished. The DIVA architecture has contributed to solving issues in the domain of software support for information visualization. Additionally, it has contributed in the field of distributed software architectures and patterns. However, there remain some open spots which are certainly interesting enough for further research. In short these issues are:

- **Collaboration**
  DIVA is a multi-user architecture with a collaborative flavor. However,

even our collaborative experiments still miss a lot of the features necessary to create an effective environment to work cooperatively. This is mainly due to the fact that it is not yet clear how people should effectively use visualization as a collaborative activity. More (empirical) research on the usefulness of communication artifacts in collaborative visualization is therefore necessary.

- **Integrated interaction**
  Although DIVA supports interaction at each phase of the visualization process, the interaction is not yet completely integrated into the visualizations. For example, to control a simulation a separate GUI component is deployed. In a completely integrated environment, the user may interact with the simulation through manipulating elements in the visualization itself. How and whether operational control should be transparently incorporated into the information visualization is open to further research.

- **2D or 3D?**
  The 2D versus 3D debate will not diminish easily. However, in the mean time, good empirical research is necessary to study the learnability, effectiveness and efficiency of 2D and 3D interfaces for both novice and experienced users.

- **Tool support**
  The DIVA project resulted in an architecture, several prototypes and a collection of Java packages. To build a new visualization application, the Java packages provide the building blocks for simulation, the Shared Concept Space and 2D or 3D visualizations. However, a certain amount of coding to glue together the application is still necessary. When a DIVA-based system would be used in a practical setting, a tool to generate visualizations (instead of coding them) would be an inevitable addition.

  As a different approach to providing tool support, the DIVA concepts could also be incorporated into an existing visualization environment such as AVS Express. DIVA components could take care of the multi-user and collaboration aspects while the commercial environment could be deployed to generate the visualizations. When such integration could be achieved, we still have all the advantages of the DIVA architecture, but combine that with the powerful tool support offered by the commercial environment.

- **Information spaces**
  The core of the DIVA architecture is the Shared Concept Space (SCS). Although the possibilities of the SCS are sufficient for our current purposes, it is a rewarding subject for further research. Possible research directions include: access control and authentication, forced structure or topology on the concepts, scalability and robustness. Furthermore, since the Shared Concept Space may be seen as a form of middleware, the

whole range of middleware research topics applies to the concept space as well.

CHAPTER 2011

The Future of Visualization

The presented ideas have absolutely no scientific foundation, they are merely based on the author's own imagination.

6.30 --- @home

It is early, too early; or probably more correct it was late, too late yesterday night. TC's head is hurting. A reminiscent pain goes from his neck, in a straight line all the way through his head and pounds in his forehead. He feels terrible and regrets the amount of drugs-experimentation he had gone through. Nevertheless he had a great time yesterday and some anti-poison will certainly make him feel better.

Yesterday, TC organized a party. Just a small party at his apartment with a couple of good friends and colleagues. He hadn't been to a party for over a year. This is mainly because he was so occupied with work. But two and a half years of working night and day have finally brought him the position he wanted so badly. Finally, he has reached the upper-management level, and he looks forward to his first day in *the eye*.

The party was a resounding success due to the large quantities of excellent drugs. Nowadays a good party abounds with all kinds of drugs, and a good host should attempt to surprise his guests with a few new inventions. Fortunately, TC has a good friend who works at the R&D-department of a large drugs producer. She had supplied TC with a couple of experimental and very strong drugs which almost take over your mental system completely. They were a big hit but now TC has to pay toll to the chemical experiments.

The number of people at the party was just right. Most of the invited guests were present and *the uninvited guests* kept quiet. TC had read some stories about a gang of young criminals who spoiled parties by intruding your house and starting fights. Therefore, TC had invited his guests only orally and not through any electronic medium. He knows exactly how unsafe public information networks are.

To wake up, TC takes a quick shower and some strong coffee. The excitement of a new episode at work certainly makes him feel better now. Shortly after a frugal breakfast he leaves for work.

7.30 --- @work

TC is nervous and feels like this is his first day at work. The rooms will open at 8 and so he has to spend another half an hour before he is allowed to enter *the eye*. He walks towards a coffee machine and selects "espresso." *Any boosts, energy, anti-stress, ginseng or vitamins added?* asks the machine in a soft computer-generated voice. *Uhhh, no thanks, I had enough of those* is TC's response.

8.00 --- @the entrance

With a loud click the door to the portal of *eye 9*, TC's own private *eye* opens. With a feeling of victory, he walks through the door and steps in front of the 'body characteristics authentication device.' He stands there for a couple of seconds until the machine says: *good morning TC, welcome to your first day in the eye*. And with that, a part of the wall moves to the side and opens the entrance to *the eye*.

8.05 --- $\in$ the eye

The eye appears to be a dome-shaped room with a diameter of about 10 metres. Exactly in the center of *the eye* stands a comfortable, black leather armchair encircled by a round desk. TC is surprised by the

emptiness of the room and wonders how he must control and manage his businesses with only a chair and a desk. He walks around the room, looking for a computer, a screen, a keyboard maybe or perhaps a connector. But his search attempts remain void.

A dim light enlightens the room. Suddenly, the wall through which TC entered *the eye* moves back to its original position. TC expects instructions and he does not have to wait long before a nice woman's voice kindly asks him to sit down:

> *Welcome to your first day in the eye, TC! I am Tina, your new secretary. To get you started in the eye, I will briefly demonstrate the capabilities of 'eye -- the dynamic collaborative management support system. Release X, Revision 1.44c.'*

TC wonders whether Tina is real or a secretary simulation agent. However, he doesn't dare to ask her that at the moment.

> *Before the instruction starts we will first go through your agenda for today.*

And with that, the ambient light dims slowly. A blurry, flickering light that seems to come from the walls starts to get stronger and stronger. After a while, TC can determine box-shaped patterns which get more focussed every second. All of a sudden, TC realizes that the complete dome is a large computer monitor.

> *Sorry for the delay in starting up the videowalls, TC. We had some problems with the new presentation devices. The drivers appeared to be buggy. Therefore, we had to go back to these slow devices. Fortunately, the image quality of the old and new devices is about the same. The old ones are only a lot slower. Anyway, our sincere apologies for any inconveniences.*

The image on the wall is completely focussed now and an extremely crisp, high-quality image of an agenda application appears. TC looks around the room and is surprised to see the agenda everywhere. He is used to his small agenda-tools on his PDA but this is something entirely different. He seems to be emerged in his agenda. Every piece of information that might be of interest is available in one or another corner of the dome.

Right in front of him is his day view. On the left a week, month and even a complete year agenda are present. To the right TC discovers a couple of still empty 'TO DO'-lists, empty mailboxes and a huge stack

of messages labeled 'management bulletin.' TC rotates 180 degrees with his comfortable armchair and sees 8 faces attentively observing him.

The faces seem to be organized in some order, but TC is not capable of immediately figuring out which layout algorithm has been used. TC glances at each of the faces and is struck by the differences. Men and women are equally present as well as different ethnic races. TC has never seen such a nice mixture of people before. Most faces are mature, wise and at the same time charismatic. While TC wonders to whom those faces belong and whether they could see him too, suddenly one of the faces moves, blinks its eyes and starts to talk.

> *Hello TC! My name is Jock, and I am currently sitting in 'eye 1,' which is across the Atlantic. All 'eyes' are scattered around the world but they are connected through an extremely fast and powerful network. By means of this interconnection we can communicate, have virtual meetings, and discuss important matters. We all know the power of contemporary computers, but we should never forget the ingenuity of cooperating intelligent human brains!*

9.06 --- ∈ the eye

Although TC has only been in *eye 9* for somewhat more than one hour, he already feels tired. It might have been yesterday's party, but the amount of new things he has to learn is also very large. Jock just explained him the history of *the eyes*. He told him how they started with a single management room about a decade ago. It was their first big experiment with visualization of management information on a large scale. Jock explained him how that old *eye* has evolved into the current networks of *eyes*. During those years they experimented with all kinds of new techniques and visualizations of which they are currently only using a small fraction.

The three most important improvements over the first *eye*, according to Jock, are the quality of the visualizations, the interaction possibilities and the means of working together with colleagues. The current visualizations provide exactly the right amount of detail, whereas the integrated interaction capabilities allow for a direct connection with the workforce. Last but not least, Jock mentioned that the communication capabilities allowed for their current conversation as well as the meeting they would have with all managers in *eyes* throughout the world, at the end of the day.

10.15 --- $\in$ the eye, $>$ coffee break

Coffee break is over and TC is sitting in the black leather chair again. While the video walls are coming back to life, TC tries to remember the program for today. He will first get a demonstration of several capabilities of the *eye*. Then there will be a demonstration of how information is presented and how the *eye* can be used to control his businesses. After that, he will meet Jock and the other faces again. Before he can recall the last thing that he is about to experience today, Tina starts to talk again:

> *Welcome back, TC! How was your coffee? We will quickly start the program now because we have got a lot to do before the meeting starts. As you can see, today's view of your agenda has been filled in while you were drinking coffee. To clean up your working environment, I will now move it to the side and make some space to show you the capabilities of the eye, Release X.*

And while Tina's words intrude TC's brain via his ears, the window containing the day view moves to his right side. As it moves, it shrinks from a large square to a 10 cm wide image. At the same time, a dome-shaped object appears in front of TC. It slowly spins around its central axis. The dome appears to be floating in space and has a dark red color. *Is this the eye?* TC asks. As if she had been waiting for this question, Tina starts explaining:

> *This is a 3D model of the eye. But before we continue, please put on your stereoscopic glasses. Only then can I take you really into every bit and piece of information about the eyes.*

TC takes his glasses out of the inside pocket of his jacket and puts them on his nose. All of a sudden, he sees the dome spinning right in front of him. He tries to touch it, but his hands move through the dome without any resistance.

> *Haha ... funny that everybody always tries to touch something although they are certain it does not exist at all ... haha ...*
>
> *Anyway, this tour starts with the technology of a single eye. After that we take a closer look at the network between the eyes. Finally, we will see how the whole thing is programmed. But let me first reassure you that this is going to be a superficial tour. We will not dive into all details underlying this extremely complex system.*

Intervention!

STOP!

I think the time has come to skip a little bit of the story that I am telling here. Not that the next part about the technical details is not interesting, on the contrary! However, I am sure you won't be able to follow any of the explanation. Or do you happen to know anything about 3^{rd} generation nano-computing, plasma-constellation projection technology, mega-bandwidth wireless networks and solar-radiation adoption encryption?

Anyway, I can imagine that you, poor readers, are now very disappointed that I am withholding you from your sneak preview into the future. Therefore, I will very briefly summarize the concepts underlying the *eye*'s technology, communication facilities and programming facilities.

A single *eye* consists of three components, a dome-shaped room, projection devices or videowalls and the processing unit or computer. The room is the easiest part. It is built out of high-quality enriched polymers which have been hardened to withstand any kind of attack. The *eye* that TC is using has a shell around the dome that has been tested with United Europe's heaviest missiles. It hardly showed any damage.

Next, we come to the presentation devices, often called videowalls after the first producer of wall attachable presentation equipment. The devices are made of a flexible, plastic-like fabric. The light emission comes from a radiating plasma that internally modifies its structure. The exact working of emitting light and absorbing energy from its environment would be getting too far off the subject. However, the quality and crispness of the equipment is orders of magnitude better than what was possible at the beginning of this century.

Finally, we come to the most complex element of a single *eye*: the core computer. And interesting enough, the biggest difference in technology can also be found in the central processing unit. During the last decade a major step in computing has been achieved. The Von Neumann principle of a CPU, memory and a bus has been left. Instead, processing power, storage and communication have been integrated into a single lump of carbon. On a scale as small as atoms, the lump of carbon --- sometimes ironically called the *brain of the eye*--- performs operations, stores information, reacts to input devices and produces output through connections to a speak synthesizer, the videowalls and possibly other output devices.

Programming the *brain* is completely different from programming a Von Neumann machine. If you want to compare it with anything that is known in the year 2K, the neurocomputer comes closest. The *brain* is not programmed by specifying an algorithm or by telling it exactly

step-by-step how it should respond to input. In contrast, the processing unit of the *eye* is trained more like you teach children. You explain the *brain* why things happen and how they happen and based on some basic knowledge the *brain* derives new knowledge. Once in a while, the new knowledge is verified with human operators or corrected by an automated knowledge correction application.

A disadvantage of the current generation of our nano-computers is that they get demented after a year. The friction within the lump of carbon produces so much heat that atoms are sometimes mutated. When the *brain* has been used for a year or so, the mutated carbon structures start to produce so many errors that the performance of the processing unit is seriously influenced. It is then time to replace the *brain* with a new, freshly trained lump of carbon.

Communication between the *eyes* takes place via a new type of wireless network. Explaining the concepts of the wireless connections is extremely complex, but believe me when I say that the response time, bandwidth and robustness are incomparable with those at the beginning of the century. Of course, the communication between the *eyes* has to be secure. And since the concept of public key encryption has been broken about 2 years ago, we are using something completely different. The workings of our new encryption mechanisms are highly classified but let me tell you that it has something to do with solar radiation at specific geographic spots on earth.

Okay, that's about all that I can tell you about the future's technology. Let's continue with the story ...

11.59 --- $\in$ eye, $\in$ canteen, $\leq$ lunch

It is almost twelve o'clock and time for lunch. Tina just finished her explanation of the technical details of the eye, its network and the security facilities. *After lunch we will focus more on how we can exploit all these technical artefacts for visualization* are the last words Tina speaks. The videowalls go back to deep black and the door through which TC entered the *eye* opens again. TC puts away his glasses, raises from his chair, stretches his back, neck and arms and walks out of the room. He passes through the entrance and walks quickly to the canteen. He is hungry.

His colleagues are already waiting for him at their lunch-meeting-point. Together they walk towards the canteen. TC picks up a tray, two glasses of milk, three sandwiches and a cheese soufflé. He pays for his lunch and walks to a table. During lunch TC and his colleagues always have interesting discussions. The subjects of their debates are diverse and range from political scandals and horrifying wars to technical issues or

interesting gossip. However, some topics seem to be more popular than others. Their favorites include complaining about the services of the system administrators, complaining about their under-estimated position in the company, and comparing technical competitors in the field of *brain* operating systems and *brain* programming environments. Whatever the subject is, however, they always try to avoid talking about their actual work.

Today the subject of their discussion seems to drift towards the new world-wide standard for mobile cyberspace. The new standard offers extremely high bandwidth wherever you are at the planet. The discussion TC and his colleagues have turns towards whether all that bandwidth is necessary.

Martine starts the discussion by stating that it is ridiculous: *Of course, mobile connectivity is fine. And the ability to exchange video and audio is cool too, but what is the use of exchanging complete DNA profiles, or exhaustive body scans?*

Security is the key, is Jack's response, *you can never be sure to whom you are talking unless you can verify the sender. The only right way to verify that is through complete information. Therefore we definitely need so much bandwidth.*

Nonsense!, Aton adds to the discussion, *When I was still young we had no mobile devices at all, let alone complete verification. When I received an email or phone call, I just had to trust it.*

Upon hearing this, Francine is triggered. *Trust, trust ... what is trust? Who can I trust? Should I trust somebody who I cannot see directly. What if it is a faker? I use mobile communication means all the time, but I have had problems with it so often, that I cannot wait until the next level of mobile security has been reached.*

I don't know what you are all talking about, Nike says, *who cares about security, authentication and trust. O my god, you are all so complicated. As long as I can watch movies on my mobile device in higher quality, I am happy.*

John J. laughs, hits with his flat hand on the table and throws his cup of soup on his white shirt.

Some things probably never change ...

13.45 --- back $\in$ the eye

TC sits in his comfortable black leather chair. He is waiting in anticipation for the visualization demonstration. Since he was a little kid, he has always been extremely visually oriented. Especially at school when

he had to memorize facts, he often drew sketches that represented and at the same time structured the information that he had to memorize. So, now that he is in the *crème de la crème* of visualization appliances he is very curious about what is coming next.

Tina briefly explains the situation he is about to experience:

> *TC, normally we would start the visualization demonstration right now. However, as you can see on the red blinking note appearing in the middle of the videowall, an emergency situation has occurred that requires your attention. Maybe, we should combine your training in using the visualization capabilities of the eye with trying to solve the problem at hand. I will try to help you as much as possible during this job. You can request my support at any time by just calling my name. However, since some tele-operation may be required during the session, please put on the tele-operation-suit now. It is available in the portal.*

TC gets out of his chair and walks to the portal. He opens the only door and there it is: an odd suit that looks like the internals of an old-fashioned computer. It is a green rubber suit with small wires going through it. TC puts it on, feels kind of stupid and walks back to the center of the eye. When he puts on his stereoscopic goggles again, he sees somebody standing in front of him.

> *Hi! My name is Thang and I am the local manager of BrainWave China. As you probably know China owns the largest brain programming factories in the world. We at BrainWave are currently experiencing some heavy competition from the NewGuys inc. Therefore, it is of highest importance that our produced brains are of outstanding quality. However, we have experienced a serious increment of complaints of bad or weak brains. The problem is that a lot of brains are programmed badly. Unfortunately, we don't know the source of the problems yet. However, we have to find it very soon, otherwise we will almost certainly loose the battle against the NewGuys.*

TC does not know very much about sales and production numbers of *brain* programming in China. Therefore, he asks Tina for a brief management overview of the situation. Within a couple of seconds, the space around TC is full of reports, charts, photographic looks into the factory and so forth. TC is amazed with the amount of available information and starts to structure the information. He discovers that he can easily grab a piece of information and move it around. When TC tries to throw a

piece of information onto another piece of information he observes a strange phenomenon. The two pieces of information merge into a single piece of information, where the two information dimensions are merged into a new perspective on the data.

After browsing through the information and trying to figure out what the source of the problem is, TC decides that the problem must be somewhere in the factory. Therefore he asks Tina whether it is possible to take a look at the factory. Tina responds that that is exactly what tele-operation is intended for. Suddenly, Mr. Thang disappears and the lights of the videowalls turn to black. A couple of seconds of complete darkness follow. Then TC is standing in the middle of a very large room. He looks around and sees a lot of desks where people are working by plugging wires in and out of things that seem to be the *brains* of modern computer systems.

> *Sorry to interrupt you again TC, but I have to tell you that you are currently virtually present in the factory in China. Currently, people there cannot see you, and you cannot interact with any of the physical objects there. This is because you are walking in a very realistic but virtual representation of the factory. The nice thing is, however, that we can modify that representation. You can project all kinds of information on any of the things available there. For example, it is possible to change the color or size of physical objects in your perspective according to a mapping that you consider valuable.*
>
> *In addition to the virtual presence mode in which you are now, you can also switch to avatar-mode. In that case an avatar robot is your presence-by-proxy that allows you to physically interact with all of the objects and people present in the factory.*

While TC walks around the factory, he notices that different types of *brain* training tables exist. Mr. Thang's explanation is that the programming tables in the factory are from three different companies to avoid dependency on a particular supplier. Of course, TC is triggered by this difference and asks Tina to colormap the errors onto the individual tables. This way, when a problem with a particular type of tables exists, he will easily discover it.

Suddenly, all tables in the factory turn into a primary color. This looks very odd, and TC is surprised. It looks as if a painter has painted all desks in the factory within a fraction of a second. Anyway, one thing is clear: the tables in ''quarter IV'' are significantly more red than the other tables. *That is impossible!* is Mr. Thang's response, *those are the newest tables, there has to be another explanation for this!*

In the next thirty minutes or so, TC and Mr. Thang examine all possible sources of the problem. All significant information dimensions, such as the raw material supplier, outside temperature, type of music being played during work and so forth are mapped onto the physical objects in the factory. This works in more or less the same way as the colored tables. However, not all information dimensions represent a physical dimension. For example, when comparing the error rate against the time of the day, it is very unnatural to map that onto a part of the factory. In those cases, a floating chart appears that contains the visualization of the relation between the investigated quantities.

Anyway, half an hour later, TC and Mr. Thang's conclusions are that the bad *brains* have nothing to do with the deployed training tables, nor with environmental influences, or the time of the day. They agree that it has to be a human thing. Therefore they deploy a colormapping to project the number of errors on the people available there. And there it is: the source of the problem. All malicious brains are being trained be a relatively small group of about 15 people, probably infiltrators from the NewGuys. Mr. Thang is very happy that they have finally found the source of the bad brains: *Thanks a lot for your assistance, now I know the problem, I will solve it as soon as possible*. TC and Mr. Thang say goodbye and TC is 'back' just in time for the meeting with Jock and his other new *eye* colleagues.

16:00 --- $\in$ the eye, the meeting

No matter how interesting the technological environment may become, most meetings remain boring ;-)

17:30 --- $\in$ the eye, the threat

Finally, the meeting is over. TC had a lot of trouble staying awake during the last hour and a half. The lack of sleep in the last night definitively made things worse. However, suddenly, TC is completely awake again. Next to him stands a long, beautiful woman that points a very cruel weapon at him. She looks angry and slowly walks towards him. TC sees that the woman changes her weapon to a somewhat smaller one. She tests her weapon, which appears to produce yellow laser beams, by shooting on the floor just in front of him. TC is scared and considers running away. However, he is paralysed with fear and remains where he is.

The woman is still approaching him and is within talking distance now. She is slightly larger than TC and she looks as if she is going to kill him within a couple of minutes. But then, TC hears Tina's voice while

the mouth of the woman is moving: *Why do you look so scared, TC? Hahaha, let's play a game of 'Threat,' the eye's version of games like Doom, Quake and Unreal. Happy fragin'!*

19:30 --- finally, @home again

TC is so exhausted, that he takes a warm bath, eats a light meal, falls asleep and dreams about how much fun his first PacMan^{tm} game was ...

Postscriptum

In case you wondered what all those strange symbols in the titles of the sections mean, I will now, probably already too late, explain what they mean:

Symbol	Etymology	Meaning in English
@	email address on Internet	at
$\in$	from mathematical set theory, meaning 'element of'	in
$>$	from mathematics and computer languages meaning 'greater than'	with respect to time: after
$\leq$	from mathematics and computer languages meaning 'less than or equal'	with respect to time: before and including

Bibliography

Advanced Visual Systems (1999), 'OpenViz: revolutionizing the display of business data'.
URL: *www.avs.com*

Ahlberg, C. & Shneiderman, B. (1994), Visual Information Seeking: tight coupling of dynamic querie filters with starfield displays, *in* 'Proceeding of CHI'94: ACM Conference on Human Factors in Computing Systems', pp. 313–317.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I. & Angel, S. (1977), *A Pattern Language*, Oxford University Press, New York.

Architecture Working Group (1999), 'Draft Recommended Practice for Architectural Description IEEE P1471/D5.2'.

Bapat, V., Drake, G. & Sadowski, D. (1998), The Arena Product Family: Enterprise Modeling Solutions, *in* 'Proceedings of the 1998 Winter Simulation Conference'.
URL: *www.sm.com/overview/whitepapers/arenafamily.htm*

Bass, L., Clements, P. & Kazman, R. (1998), *Software Architecture in Practice*, SEI series in Software Engineering, Addison-Wesley Publishing Company.

Baumann, J., Hohl, F., Rothermel, K., Schwehm, M. & Strasser, M. (1998), Mole 3.0: A Middleware for Java-based Mobile Software Agents, *in* 'Proceedings of Middleware'98', pp. 355–370.

Bentley, R., Rodden, T., Sawyer, P. & Sommerville, I. (1994), 'Architectural support for cooperative multiuser interfaces', *IEEE Computer* **27**(5), 37–46.

Bertin, J. (1977/1981), *Graphics and Graphic Information Processing*, De Gruyter.

Blaxxun Interactive (1997), 'www.blaxxun.com'.
URL: *www.blaxxun.com*

Bolier, D. & Eliëns, A. (1994), Sim — a C++ library for discrete event simulation, Technical Report IR-367, Vrije Universiteit, Amsterdam.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. & Stal, M. (1996), *Pattern-oriented software architecture: a system of patterns*, John Wiley & Sons.

Card, S. K., Mackinlay, J. D. & Shneiderman, B. (1999), *Readings in Information Visualization: using vision to think*, Morgan Kaufmann.

Clements, P. C. (1996), Coming Attractions in Sofware Architectures, Technical report, Software Engineering Institute.

Dastani, M. M. (1998), Languages of Perception, PhD thesis, Universiteit van Amsterdam.

Derthick, M., Kolojejchick, J. & Roth, S. F. (1997), An interactive visualization environment for data exploration, *in* 'Proceedings of Knowledge Discovery in Databases 1997', AAAI Press, pp. 2–9.

Dijkstra, E. (1968), 'The structure of the THE-multiprogramming system', *Communications of the ACM* **11**(5), 341–346.

D'Souza, D. F. & Wills, A. C. (1999), *Objects, Components, and Frameworks with UML: The Catalysis Approach*, Addison-Wesley Publishing Company.

Earnshaw, R. & Vince, J. (1999), *Digital Convergence: the Information Revolution*, Springer Verlag.

Eliëns, A., Niessink, F., Schönhage, B., van Ossenbruggen, J. & Nash, P. (1996), Support for Business Process Redesign: Simulation, Hypermedia and the Web, *in* 'Euromedia 96: Telematics in a Multimedia Environment, London, United Kingdom', The Society for Computer Simulation International, pp. 193–200.

Eliëns, A., van Ossenbruggen, J. & Schönhage, B. (1997), Animating the Web — An SGML-based Approach, *in* R. Earnshaw & J. Vince, eds, 'The Internet in 3D — Information, Images and Interaction', Academic Press.

Ellis, C., Gibbs, S. & Rein, G. (1991), 'Groupware: some issues and experiences', *Communications of the ACM* **34**(1), 680–689.

Elvins, T. T. & Johnson, G. (1998), 'Introduction to Collaborative Visualization', *Siggraph computer graphics* **32**(2).
URL: *siggraph.org/publications/newsletter/v32n2/*

Fowler, M. (1997), *UML distilled: applying the standard object modeling language*, Addison-Wesley Publishing Company.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994), *Design Patterns — Elements of Reusable Object-Oriented Software*, Professional Computing Series, Addison-Wesley Publishing Company.

Garlan, D. & Perry, D. E. (1995), 'Introduction to the Special Issue on Software Architecture', *IEEE Transactions on Software Engineering* **21**(4), 269–274.

Gerrits, J. (1995), Towards Information Logistics: An Exploratory Study of Logistics in Information Production, PhD thesis, Vrije Universiteit, Amsterdam, Faculty of Economic Sciences, Business Administration and Econometrics.

Gershon, N., Eick, S. G. & Wright, W. (1997), 'Information Visualization Applications in the Real World: business visualization applications', *IEEE Computer Graphics & Applications* **17**(4), 66–70.

ISO (1997), *The Virtual Reality Modeling Language*. International Standard ISO/IEC IS 14772-1:1997.

Kazman, R. & Carriere, J. (1996*a*), An adaptable software architecture for rapidly creating information visualizations, *in* 'Proceedings of Graphics Interface '96', pp. 17–27.

Kazman, R. & Carriere, J. (1996*b*), Rapid prototyping of information visualization using VANISH, *in* 'Proceedings of InfoVis '96', pp. 21–28.

Koike, H. & Yoshihara, H. (1993), Fractal approaches for visualizing huge hierarchies, *in* 'Proceedings of the IEEE 1993 symposium in visual languages', pp. 55–60.
URL: *www.vogue.is.uec.ac.jp/~koike/papers/vl93/vl93.html*

Kolojejchick, J., Roth, S. F. & Lucas, P. (1997), 'Information appliances and tools in Visage', *IEEE Computer Graphics & Applications* **17**(4), 32–41.

Mackinlay, J. (1986), 'Automating the design of graphical representations of relational information', *ACM Transactions on Graphics* **5**(2), 110–141.

Martin, J. (1991), *Rapid Application Development*, MacMillan.

McCormick, B., DeFanti, T. & Brown, M. (1987), 'Visualization in Scientific Computing', *ACM SIGGRAPH Computer Graphics* **21**(6).

Microsoft (1998), 'DCOM Architecture - white paper'.
URL: *www.microsoft.com/com/*

Microsoft (1999), *Microsoft Excel 2000 Step by Step*, Microsoft Press.

Miller, J. A., Ge, Y. & Tao, J. (1998), Component-based Simulation Environments: JSIM as a case study using java beans, *in* 'Proceedings of the 1998 Winter Simulation Conference (WSC'98), Washington, DC', pp. 373–381.

Moltenbrey, K. (1999), 'Going the extra mile', *Computer Graphics World* **22**(9).

Mowbray, T. J. & Malveau, R. C. (1997), *CORBA design patterns*, John Wiley & Sons.

Nasa Goddard Space Flight Center (2000), 'Visual Analysis Graphical Environment'.
URL: *tidalwave.gsfc.nasa.gov/avatar/visage*

Nielsen, J. (1998), '2D is better than 3D'.
URL: *www.zdnet.com/devhead/alertbox/981115.html*

ObjectSpace (1998), *Voyager Core Technology 2.0 User Guide*.
URL: *www.objectspace.com/products/voyager/*

Parnas, D., Clements, P. & Weiss, D. (1985), 'The modular structure of complex systems', *IEEE Transactions on Software Engineering* **11**(3), 259–266.

Pesce, M. (1995), *VRML: Browsing and Building Cyberspace*, New Riders Publishing.

Plaisant, C., Milash, B., Rose, A., Widoff, S. & Shneiderman, B. (1996), LifeLines:Visualizing Personal Histories, *in* 'Proceedings of CHI'96, ACM Conference on Human Factors in Computing Systems', pp. 221–227.

Platinum (1998), 'Putting metadata to work in the warehouse (white paper)'. available via: www.platinum.com.

Reinhard, W., Schweizer, J. & Völksen, G. (1994), 'CSCW Tools: concepts and architectures', *IEEE Computer* **27**(5), 28–36.

Robertson, G., Card, S. & Mackinlay, J. (1993), 'Information Visualization using 3D Interactive Animation', *Communications of the ACM* **36**(4), 57–71.

Robertson, G. G., Mackinlay, J. D. & Card, S. K. (1991), Cone Trees: animated 3D visualizations of hierarchical information, *in* 'Proceedings of the ACM SIGCHI 1991 Conference on Human Factors in Computing Systems', pp. 189–194.

Rumbaugh, J., Jacobson, I. & Booch, G. (1999), *The Unified Modeling Reference Manual*, Addison-Wesley Publishing Company.

Schmidt, D., Stal, M., Rohnert, H. & Buschmann, F. (2000), *Pattern-oriented software architecture, volume 2: patterns for concurrent and distributed objects*, John Wiley & Sons.

Schönhage, B., Bakker, P. & Eliëns, A. (1998), So Many Users — So Many Perspectives, *in* 'Proceedings of "Designing effective and usable multimedia systems", 9-10 September 1998, Fraunhofer Institute IAO, Stuttgart, Germany', IFIP.

Schönhage, B. & Eliëns, A. (1997), A flexible architecture for user-adaptable visualization, *in* D. S. Ebert & C. K. Nicholas, eds, 'Workshop on New Paradigms in Information Visualization and Manipulation '97, Conference on Information and Knowledge Management, 10 - 14 November 1997, Las Vegas, USA', ACM Press.

Schönhage, B. & Eliëns, A. (1998), Multi-user Visualization: a CORBA/Web-based approach, *in* 'Proceedings of "Digital Convergence: the Future of the Internet and WWW", 20-23 April 1998, Bradford, United Kingdom', British Computer Society.

Schönhage, B. & Eliëns, A. (1999*a*), Dynamic and Mobile VRML gadgets, *in* 'Proceedings of VRML99- International Conference on the Virtual Reality Modeling Language and Web3D technologies'.

Schönhage, B. & Eliëns, A. (1999*b*), Four Ways to Architect your Distributed Objects, *in* J. Bosch, ed., 'Proceedings of second Nordic Workshop on Software Architecture'.

Schönhage, B. & Eliëns, A. (1999*c*), From Distributed Object Features to Architectual Styles, *in* 'Proceedings of Engineering Distributed Object 99, ICSE 99 workshop', pp. 48–55.

Schönhage, B. & Eliëns, A. (2000*a*), Information Exchange in a Distributed Visualization Architecture: the Shared Concept Space, *in* 'Proceedings of Distributed Objects and Applications 2000', IEEE Computer Society Press. To be published.

Schönhage, B. & Eliëns, A. (2000*b*), Management through Vision: a case study towards requirements of BizViz, *in* 'Information Visualization 2000 (IV2000) - London, England', IEEE Computer Society Press. To be published.

Schönhage, B., Eliëns, A. & van Ballegooij, A. (2000), 3D Gadgets for Business Process Visualization: a case study, *in* 'Proceedings of Web3D/VRML2000 - International Conference on the Virtual Reality Modeling Language and Web3D technologies', pp. 131—138, 174.

Schroeder, W. J., Avilla, L. S. & Hoffman, W. (2000), 'Visualizing with VTK: a tutorial', *IEEE Computer Graphics & Applications* **20**(5), 20–27.

Schroeder, W., Martin, K. & Lorensen, B. (1996), *The Visualization Toolkit: an Object-Oriented Approach to 3D Graphics*, Prentice Hall.

Sebrechts, M. M., Vasilakis, J., Miller, M. S., Cugini, J. V. & Laskowski, S. J. (1999), Visualization of Search Results: A Comparative Evaluation of Text, 2D and 3D Interfaces, *in* 'Proceedings of 22nd ACM SIGIR conference on Research and development in information retrieval', pp. 3–10.

Shaw, M. & Clements, P. (1997), A Field Guide to Boxology: Preliminary classification of architectural styles for software systems, *in* 'Proceedings of COMPSAC, Washington, D.C.'.

Shaw, M. & Garlan, D. (1996), *Software Architecture: perspectives on an emerging discipline*, Prentice Hall.

Shneiderman, B. (1994), 'Dynamic Queries for Visual Information Seeking', *IEEE Software* **11**(6), 70–77.

Shneiderman, B. (1996), The Eyes Have It: a task by data type taxonomy for information visualization, *in* 'Proceedings of IEEE Workshop on Visual Languages', pp. 336–343.

Shneiderman, B. (1998), *Designing the User-Interface, Strategies for Effective Human-Computer Interaction*, 3rd edn, Addison-Wesley Publishing Company.

Siegel, J. (1996), *CORBA Fundamentals and Programming*, John Wiley & Sons.

Sowizral, H., Rushforth, K. & Deering, M. (1997), *The Java 3D Api Specification (Java Series)*, Addison-Wesley Publishing Company.

Sunsoft (1998), 'Java Remote Method Invocation - white paper'.
URL: *java.sun.com/docs/white/*

Tanenbaum, A. S. (1995), *Distributed Operating Systems*, Prentice Hall.

Tufte, E. R. (1983), *The visual display of quatitative information*, Graphics Press.

Tufte, E. R. (1990), *Envisioning Information*, Graphics Press.

Tufte, E. R. (1997), *Visual explanations: images and quantities, evidence and narrative*, Graphics Press.

Turban, E. & Aronson, J. E. (1998), *Decision support systems and intelligent systems*, Prentice Hall.

van Liere, R., Harkes, J. & de Leeuw, W. (1998), A Distributed Blackboard Architecture for Interactive Data Visualizaiton, *in* D. Ebert, H. Rushmeier & H. Hagen, eds, 'Proceedings of IEEE Visualization'98 Conference', IEEE Computer Society Press.

Walker, G. (1995), 'Challenges of Information Visualization', *British Telecommunications Engineering* **14**(April), 17–25.

Ware, C. (2000), *Information Visualization: perception for design*, Morgan Kaufmann.

Westmacott, I. (1997), 'Advanced Visual Systems AVS/Express 3.0', *Sun Expert* .
URL: *www.avs.com/company/articles/sunexpert/*

Wright, W. (1995), Information Animation Applications in the Capital Markets, *in* 'Proceedings of InfoVis'95, IEEE Symposium on Information Visualization', pp. 29–25.

Zhang, P. (1996), 'Visualizing Production Planning Data', *IEEE Computer Graphics & Applications* **16**(5), 7–10.

Nederlandstalige samenvatting

Diva: Architecturale Perspectieven op Informatie Visualisatie

Informatie visualisatie is de presentatie van informatie door middel van plaatjes. Maar visualisatie is meer dan alleen het visueel weergeven van gegevens. Informatie visualisatie wordt altijd ingezet met een doel. Eén doel voor het gebruik van visualisatie is om gegevens te *begrijpen*, of zoals (Card et al. 1999) het noemen: *om je cognitie te vergroten* (to amplify cognition). Een tweede doel van visualisatie is om informatie *uit te wisselen* door middel van de menselijke visuele capaciteiten.

Computer ondersteuning voor visualisatie was altijd gericht op wetenschappelijke visualisatie. Gedurende de laatste jaren heeft informatie visualisatie echter steeds meer interesse weten te winnen. Dit heeft geresulteerd in zowel tools die speciaal gericht zijn op informatie visualisatie als in ingebouwde visualisatie ondersteuning voor bestaande gegevens beheer tools.

Problemen met de huidige generatie van tools ter ondersteuning van informatie visualisatie zijn drieledig. Ten eerste zijn de meeste tools gericht op een enkele gebruiker terwijl visualisatie, zeker als het gebruikt wordt om problemen op te lossen of beslissingen te nemen, een groepsactiviteit is. Ten tweede hebben veel tools wel een ingebouwde mogelijkheid om hun eigen data te visualiseren. Maar het importeren van gegevens uit andere bronnen is dan niet mogelijk. Tenslotte staat interactie in visualisatie omgevingen nog in de kinderschoenen. Het is gewoonlijk wel mogelijk om met het gegenereerde visuele model te interacteren, maar interactief de gegevensbron of visuele *mapping* te veranderen is onmogelijk.

Een specifiek domain in het gebied van de informatie visualisatie is bedrijfsvisualisatie of *BizViz*. Het doel van *BizViz* is om beslissingsmakers te ondersteunen door middel van snel toegankelijk informatie van hoge kwaliteit. In een

case study bij ASZ / Gak Nederland hebben we laten zien hoe *BizViz* managers kan helpen bij het begrijpen van complexe informatie wat vervolgens leidde tot betere beslissingen.

Visualisatie modellen beschrijven visualisatie als een proces dat gegevens in plaatjes converteert. Het visualisatie proces bestaat uit een reeks transformatie stappen. Het visualisatie referentie model, bijvoorbeeld, deelt het visualisatie proces op in vier types van gegevens: ruwe data, gegevenstabellen, visuele structuren en *views* (kijk of blik). De overgangen tussen deze vier gegevenstypes vormen samen de totale afbeelding van ruwe data tot visuele entiteit. Menselijke interactie, die expliciet aanwezig is in het referentie model, kan plaats vinden tijdens alle drie de overgangen tussen de verschillende gegevenstypes.

DIVA, de gedistribueerde visualisatie architectuur (*Distributed Visualization Architecture*) beschrijft een visualisatie architectuur die bedoelt is voor interactieve, meerdere-gebruikers visualisatie in een gedistribueerde omgeving. Oftewel, het pakt de problemen met de tegenwoordige visualisatie tools aan en breidt het visualisatie referentiemodel uit naar het domein van meerdere gebruikers en meerdere perspectieven.

In dit proefschrift wordt een gedistribueerde visualisatie architectuur beschreven volgens drie architecturele perspectieven: de conceptuele architectuur, de software architectuur en de informatie architectuur. De belangrijkste rol wordt gespeeld door de software architectuur die de componenten waaruit de architectuur bestaat beschrijft. Verder beschrijft de software architectuur de relaties en interactie tussen de componenten.

In essentie bestaat de DIVA architectuur uit een informatie voorziener, een ontkoppeld gegevens model en presentatie componenten. De ontkoppeling van de generatie van informatie, de verwerking en tenslotte de presentatie geeft de DIVA architectuur de flexibiliteit die het nodig heeft om de eis om meerdere gebruikers te ondersteunen waar te maken.

In het DIVA project heeft de praktische kant van software engineering steeds een belangrijke rol gespeeld. Daarom zijn er ook vier *case studies* uitgevoerd die zich hebben gericht op verschillende problemen. Bijvoorbeeld, *The Great Dictator* was een *case study* met als doel een collaboratieve visualisatie omgeving te realiseren. Het prototype dat het resultaat was van deze exercitie is gebaseerd op de standaard DIVA software architectuur maar bevat tevens collaboratie componenten om de samenwerking tussen de participanten mogelijk te maken. De technologie die voor dit prototype gebruikt werd is gebaseerd op gedistribueerde en mobiele objecten. De visualisaties zelf worden getoond in de 3D modelleer taal VRML.

Een andere technologie voor het presenteren van 3D visualisaties is Java3D. Java3D is dan ook gebruikt in een 3D versie van de Gak bedrijfsproces visualisaties. Daarvoor is een kleine, herbruikbare verzameling van 3D *gadgets* ontwikkeld die gemakkelijk in het al bestaande systeem gestopt kon worden. Hierdoor konden we experimenteren met de visualisatie van management informatie in drie dimensies. Een uitputtende evaluatie van de 3D visualisaties

was niet mogelijk. Wel bleek dat het belangrijkste verschil tussen de 2D en 3D versie was dat de 2D visualisatie veel toegankelijker was terwijl de 3D visualisatie meer informatie kon bevatten, maar dit wel tegen de prijs van toegenomen complexiteit.

DIVA is een gedistribueerde architectuur. Het laatste deel van dit proefschrift is daarom gewijd aan gedistribueerde software architecturen. Of iets meer precies, het laatste deel focust op informatie uitwisseling in gedistribueerde omgevingen. In DIVA is de gedeelde concept ruimte, de *Shared Concept Space (SCS)*, het gebruikte model voor de informatie uitwisseling tussen informatie bronnen en visualisatie componenten. De SCS is gebaseerd op drie bekende ontwerp patronen (*design patterns*) namelijk *Blackboard*, *Model-View-Controller* en *Talker-Listener*. De SCS heeft van alle drie de patronen eclectisch bepaalde delen overgenomen. Een verschil is echter dat de SCS nieuwe informatie onafhankelijk van de informatie voorziener of viewers kan afleiden. Op deze manier kan kennis omtrent de afleiding van nieuwe informatie gedeeld en hergebruikt worden door verschillende componenten en mensen.

Architectuur stijlen zijn patronen die component types, executie controle en gegevens uitwisseling op software architectuur niveau beschrijven. Gebaseerd op onze ervaring met DIVA hebben we vier architectuur stijlen geïntroduceerd om gedistribueerde, object-georienteerde software te beschrijven: de *distributed objects*, *dynamically downloaded classes*, *mobile objects* en *event-space* architectuur stijlen. De stijlen verschillen in object eigenschappen, connectie types, en locatie eigenschappen. Dit betekent dat ze, afhankelijk van de eisen aan de architectuur, in verschillende contexten ingezet kunnen worden. De besproken vuistregels zijn een eerste stap in de richting van de keuze voor één specifieke stijl in bepaalde situaties.

Samenvattend beschrijft dit proefschrift een architectuur voor dynamische, interactieve visualisatie die meerdere gebruikers en meerdere perspectieven ondersteunt. De drie hoofdproblemen van huidige visualisatie tools, namelijk weinig ondersteuning voor meerdere gebruikers, nauwe koppeling (*tight coupling*) en beperkte interactiemogelijkheden vormden de uitgangspunten voor DIVA. Concluderend heeft dit project zeker bijgedragen om de problemen op te lossen. De DIVA architectuur ondersteunt namelijk meerdere personen en perspectieven en heeft door middel van de SHARED CONCEPT SPACE een losgekoppelde (maar toch geïntegreerde) structuur. Tenslotte maakt DIVA interactieve visualisatie mogelijk: alle aspecten van het visualisatie proces kunnen worden beïnvloed om zo tot een nuttige en waardevolle visualisatie te komen.

Titles in the SIKS Dissertation Series

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects
Promotor: Prof.dr. M.L. Kersten (CWI/UvA)
Co-promotor: dr. A.P.J.M. Siebes (CWI)
Promotie: 30 maart 1998

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information
Promotores: Prof.dr.ir. A. Hasman (UM)
Prof.dr. H.J. van den Herik (UM/RUL)
Prof.dr.ir. J.L.G. Dietz (TUD)
Promotie: 7 mei 1998

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective
Promotores: Prof.dr.ir. J.L.G. Dietz (TUD)
prof.dr. P.C. Hengeveld (UvA)
Promotie: 22 juni 1998

1998-4 Dennis Breuker (UM)
Memory versus Search in Games
Promotor: Prof.dr. H.J. van den Herik (UM/RUL)
Promotie: 16 oktober 1998

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting
Promotores: Prof.mr. H. Franken
Prof.dr. H.J. van den Herik
Promotie: 13 mei 1998

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products
Promotor: prof.dr. J. Treur
Co-promotor: Dr.ir. M. Willems
Promotie: 11 mei 1999

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets
Promotor: prof. dr. A. de Bruin
Co-promotor: Dr. J.C. Bioch
Promotie: 4 juni 1999

1999-3 Don Beal (Queen Mary and Westfield College)
The Nature of Minimax Search
Promotor: Prof.dr. H.J.van den Herik
Promotie: 11 juni 1999

1999-4 Jacques Penders (KPN Research)
The practical Art of Moving Physical Objects
Promotor: Prof.dr. H.J. van den Herik
Co-promotor: Dr. P.J. Braspenning
Promotie: 11 juni 1999

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems
Promotor: Prof.Dr. R.A. Meersman
Co-promotor: Dr. H. Weigand
Promotie: 1 oktober 1999

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems
Promotor: prof.dr. J. Treur
Copromotor: Dr. F.M.T. Brazier
Promotie: 30 september 1999

1999-7 David Spelt (UT)
Verification support for object database design
Promotor: Prof. Dr. P.M.G. Apers
Assistent promotor: Dr. H. Balsters
Promotie: 10 september 1999

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation
Promotor: Prof. dr. H.J. van den Herik
Co-promotor: Dr. P.J. Braspenning
Promotie: 3 december 1999

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance
Promotor: prof.dr. J.C. van Vliet (VU)
Promotiedatum: 28 maart 2000

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management
Promotores: prof. dr. P.M.E. De Bra
prof. dr. R.H. McClatchey
Copromotor: dr. P.D.V. van der Stok
Promotie: 29 mei 2000

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief
Promotor: Prof. dr. B.J. Wielinga
Co-promotor: Dr. P.A.A. van den Besselaar
Promotie: 20 juni 2000

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design
Promotor: Prof. dr. J.C. van Vliet
Co-promotores: Dr. G.C. van der Veer
Dr. M.J. Tauber
Promotie: 10 oktober 2000

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval
Promotores: Prof.dr. H.J. van den Herik (UM/RUL)
Prof.dr.ir. J.L.G. Dietz (TUD)
Prof.dr.ir. A. Hasman (UM)
Promotie: 14 september 2000

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication
Promotor: prof. dr. John-Jules Ch. Meyer
Co-promotoren: Dr. Frank S. de Boer
Dr. Wiebe van der Hoek
Promotie: 18 oktober 2000

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management
Promotor: prof.dr. J.-J. Ch. Meyer
Co-promotor: Dr.P.J.F.Lucas
Promotie: 30 oktober 2000

2000-8 Veerle Coupé (EUR)
Sensitivity Analyis of Decision-Theoretic Networks
Promotores: prof.dr.J.D.F. Habbema

		prof.dr.ir.L.C van der Gaag
	Promotie:	27 september 2000

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization
Promotor: prof. dr. M.L. Kersten (CWI/UvA)
Promotie: 03 november 2000

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture
Promotor: prof. dr. M.L. Kersten (CWI/UvA)
Promotie: 14 december 2000

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management
Promotor: prof. dr. M.L. Kersten (CWI/UvA)
Promotie: 14 december 2000

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks
Promotores: prof.dr. J.-J.Ch. Meyer (UU)
prof.dr.ir. L.C. van der Gaag (UU)
Co-promotor: dr. C.L.M Witteman (UU)
Promotie: 12 maart 2001

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models
Promotor: prof. dr. J.-J.Ch. Meyer (UU)
Co-Promotoren: dr. W. van der Hoek (UU)
dr. F.S. de Boer (UU)
Promotie: 5 februari 2001

2001-3 Maarten van Someren (UvA)
Learning as problem solving
Promotor: prof. dr. B.J. Wielinga (UvA)
Promotie: 1 maart 2001

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets
Promotor: Prof. dr. H.J. van den Herik (UM/RUL)
Promotie: 22 februari 2001

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style
Promotor: prof.dr. J.C. van Vliet (VU)
Promotie: 10 april 2001

2001-6 Martijn van Welie (VU)
Task-based User Interface Design

	Promotor:	prof.dr. J.C. van Vliet (VU)
	Co-Promotoren:	dr. G.C. van der Veer (VU)
		dr. A. Eliëns (VU)
	Promotie:	17 april 2001

2001-7 Bastiaan Schönhage (VU)
Diva: Architectural Perspectives on Information Visualization

Promotor:	prof.dr. J.C. van Vliet (VU)
Co-Promotor:	dr. A. Eliëns (VU)
Promotie:	8 mei 2001

2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics

Promotores:	prof.dr. F.M.T. Brazier (VU)
	prof.dr. J. Treur (VU)
Promotie:	12 juni 2001

Index